Scientific Annals of Computer Science vol. 35 (2), 2025, pp. 211–249
doi: 10.47743/SACS.2025.2.211

Formalizing the Notions of Non-Interactive and Interactive Algorithms

C.A. Middelburg [1]

Abstract

An earlier paper gives an account of a quest for a satisfactory formalization of the classical informal notion of an algorithm. That notion only covers algorithms that are deterministic and non-interactive. In this paper, an attempt is made to generalize the results of that quest first to a notion of an algorithm that covers both deterministic and non-deterministic algorithms that are non-interactive and then further to a notion of an algorithm that covers both deterministic and non-deterministic algorithms that are interactive. The notions of an non-interactive proto-algorithm and an interactive proto-algorithm are introduced. Non-interactive algorithms and interactive algorithms are expected to be equivalence classes of non-interactive proto-algorithms and interactive proto-algorithms, respectively, under an appropriate equivalence relation. On both non-interactive proto-algorithms and interactive proto-algorithms, three equivalence relations are defined. Two of them are deemed to be bounds for an appropriate equivalence relation and the third is likely an appropriate one.

Keywords: non-interactive algorithm, interactive algorithm, proto-algorithm, algorithmic equivalence, computational equivalence, non-determinism

This work is licensed under the Creative Commons Attribution Licence (CC BY)

[1]Informatics Institute, Faculty of Science, University of Amsterdam, Science Park 900, 1098 XH Amsterdam, the Netherlands, Email: C.A.Middelburg@uva.nl

1 Introduction

In [11] an account is given of a quest for a satisfactory formalization of the notion of an algorithm that is informally characterized in standard works from the mathematical and computer science literature such as [8, 9, 10, 12]. The notion of an algorithm in question is generally considered the original notion of an algorithm. Because several generalizations of this notion have emerged, it is now often referred to as the notion of a classical algorithm. Classical algorithms are known for their deterministic and non-interactive nature.

Non-deterministic algorithms, introduced in [5], are widely used as a starting point for developing deterministic backtracking algorithms. Moreover, they play an important role in the field of computational complexity. In this paper, an attempt is made to generalize the results of the above-mentioned quest to a notion of an algorithm that covers both deterministic and non-deterministic algorithms that are non-interactive. The term non-interactive algorithm is coined for this notion of an algorithm. Interactive algorithms, which emerged due to the advent of interactive computation, are currently widespread. In this paper, an attempt is made to generalize the results of the above-mentioned quest further to a notion of an algorithm that covers both deterministic and non-deterministic algorithms that are interactive.

Interactive algorithms are mainly described with sentences like "An interactive algorithm is an algorithm that can interact with the environment in which it is applied". However, there is no consensus on which properties characterize interactive algorithms well. In [2], a specific view on the nature of interactive algorithms is discussed in detail, culminating in a characterization of interactive algorithms by a number of postulates. The view concerned is the only one found in the computer science literature so far. Some of its details are based on choices whose impact on its generality is not clear.

Despite the fact that interactive algorithms are currently widespread, no other work aimed at a satisfactory formalization of the notion of an interactive algorithm has been reported in the computer science literature. This state of affairs motivated me to start a quest for a satisfactory formalization of the notion of an interactive algorithm. The main goals of this quest are to provide a framework for studying complexity-theoretic issues concerning interactive algorithms and to provide a semantic basis for languages to describe interactive algorithms.

Due to the advent of interactive computation, several related models of interactive computation have been proposed. They are based on variants of

Turing machines, to wit interactive Turing machines [13], persistent Turing machines [6], and reactive Turing machines [1]. In this paper, the starting point for the formalization of the notion of an interactive algorithm is a characterization of the notion by properties suggested by those models of interactive computation.

In [11], first the notion of a classical proto-algorithm is introduced and then three equivalence relations on classical proto-algorithms are defined. The thought is that classical algorithms are equivalence classes of classical proto-algorithms under an appropriate equivalence relation. Two of the three equivalence relations defined give bounds for an appropriate equivalence relation and the third one is likely an appropriate one. In this paper, the same approach is followed for non-interactive algorithms and interactive algorithms.

This paper is organized as follows. First, the basic notions and notations used in this paper are introduced (Section 2). Next, intuitive characterizations of the notion of a non-interactive algorithm and the notion of an interactive algorithm are given by properties that are considered to belong to the most important ones of these notions (Section 3). After that, the formal notion of a non-interactive proto-algorithm is introduced and three relevant equivalence relations on non-interactive proto-algorithms are defined (Section 4). Then, the formal notion of an interactive proto-algorithm is introduced and three relevant equivalence relations on interactive proto-algorithms are defined (Section 5). Thereafter, the connection between non-interactive proto-algorithms and interactive proto-algorithms is addressed (Section 6). Finally, some concluding remarks are made (Section 7).

In this paper, an attempt is made to generalize the work on the classical notion of an algorithm presented in [11]. In Sections 3 and 4, this has led to text overlap with [11].

2 Preliminaries

In this section, the basic notions and notations used in this paper are introduced.

The notion of a non-interactive proto-algorithm and the notion of an interactive proto-algorithm will be formally defined in Sections 4.1 and 5.1, respectively, in terms of three auxiliary notions. The definition of one of the auxiliary notions is based on the well-known notion of a rooted labeled directed graph. However, the definitions of this notion given in the

mathematical and computer science literature vary. Therefore, the definition that is used in this paper is given.

Definition 1 *A* rooted labeled directed graph *G is a sextuple $(V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, where:*

- *V is a non-empty finite set, whose members are called the* vertices *of G;*
- *E is a subset of $V \times V$, whose members are called the* edges *of G;*
- *L_v is a countable set, whose members are called the* vertex labels *of G;*
- *L_e is a countable set, whose members are called the* edge labels *of G;*
- *l is a partial function from $V \cup E$ to $L_\mathrm{v} \cup L_\mathrm{e}$ such that*

  for all $v \in V$ for which $l(v)$ is defined, $l(v) \in L_\mathrm{v}$ and
  for all $e \in E$ for which $l(e)$ is defined, $l(e) \in L_\mathrm{e}$,

  called the labeling function *of G;*
- *$r \in V$, called the* root *of G.*

The additional graph theoretical notions defined below are also used.

Definition 2 *Let $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$ be a rooted labeled directed graph. Then a* cycle *in G is a sequence $v_1 \ldots v_{n+1} \in V^*$ such that, for all $i \in \{1, \ldots, n\}$, $(v_i, v_{i+1}) \in E$, $\mathrm{card}(\{v_1, \ldots, v_n\}) = n$, and $v_1 = v_{n+1}$. Let, moreover, $v \in V$. Then the* indegree *of v, written* $\mathrm{indeg}(v)$*, is* $\mathrm{card}(\{v' \mid (v', v) \in E\})$ *and the* outdegree *of v, written* $\mathrm{outdeg}(v)$*, is* $\mathrm{card}(\{v' \mid (v, v') \in E\})$.

Concerning sequences, we write:

- $\mathcal{A}^\infty$ for the set of all non-empty sequences over the set $\mathcal{A}$ that are finite or countably infinite;
- $\langle\rangle$ for the empty sequence;
- $\langle a \rangle$ for the sequence having a as sole element;
- $\alpha \frown \alpha'$, for the concatenation of the sequences α and α';
- $\alpha[n]$ for the nth element of the sequence α if there exists a prefix of α of length n and otherwise the last element of α;

- $|\alpha|$ for the length of the sequence α.

Concerning tuples, we write:

- $\boldsymbol{a}(i)$, where $\boldsymbol{a} = (a_1, \ldots, a_n)$, for a_i;
- $\boldsymbol{a}[i \mapsto a]$, where $\boldsymbol{a} = (a_1, \ldots, a_n)$, for $(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n)$.

Moreover, we write:

- $\mathbb{N}^+$ for the set $\{n \in \mathbb{N} \mid n > 0\}$ of positive natural numbers;
- $\mathcal{A}^n$ for the n-fold Cartesian product of the set $\mathcal{A}$ with itself;
- $\mathcal{P}(\mathcal{A})$ for the set of all subsets of the set $\mathcal{A}$.

In Sections 4 and 5, we assume the existence of a dummy value $\perp$ that is not a member of certain sets. Concerning $\perp$, we write:

- $\mathcal{A}_\perp$, where $\mathcal{A}$ is a set such that $\perp \notin \mathcal{A}$, for $\mathcal{A} \cup \{\perp\}$;
- $\perp^n$ for the unique member of the set $\{\perp\}^n$.

In this paper, we sometimes consider multi-valued functions, i.e. functions from a set $\mathcal{A}$ to the set of all subsets of a set $\mathcal{A}'$. The following notion concerning multi-valued functions is used:

Definition 3 *Let $\mathcal{A}$ and $\mathcal{A}'$ be sets, and let f and g be functions from $\mathcal{A}$ to $\mathcal{P}(\mathcal{A}')$. Then f is* smaller then *g if $f(s) \subset g(s)$ for all $s \in \mathcal{A}$.*

Moreover, the following notion of a finitely generated set is used:

Definition 4 *Let $\mathcal{A}$ be a set. Then $\mathcal{A}$ is a* finitely generated set *if there exist a finite subset $\mathcal{A}'$ of $\mathcal{A}$ and a finite set $\mathcal{F}$ of functions on $\mathcal{A}$ such that, for each subset $\mathcal{A}''$ of $\mathcal{A}$ that includes $\mathcal{A}'$, the closure of $\mathcal{A}''$ under the functions in $\mathcal{F}$ equals $\mathcal{A}$.*

3 Informal Notions of Algorithm

In this paper, the notion of a classical algorithm concerns the original notion of an algorithm, i.e. the notion that is intuitively characterized in standard works from the mathematical and computer science literature such as [8, 9, 10, 12]. Seeing the terminology used for proposed generalizations

of the original notion of an algorithm, a classical algorithm could also be called a deterministic non-interactive algorithm.

The characterizations of the notion of a classical algorithm referred to above indicate that a classical algorithm is considered to express a pattern of behaviour by which all instances of a computational problem can be solved. This calls for a description of a classical computational problem that does not refer to the notion of a classical algorithm:

> A classical computational problem is a problem where, given a value from a certain set of input values, a value from a certain set of output values that is in a certain relation to the given input value must be produced if it exists. The input values are also called the instances of the problem and an output value that is in the certain relation to a given input value is also called a solution for the instance concerned.

From the existing viewpoints on what a classical algorithm is, it follows that the following properties must be considered the most important ones of a classical algorithm:

1. a classical algorithm is a finite expression of a pattern of behaviour by which all instances of a classical computational problem can be solved;
2. the pattern of behaviour expressed by a classical algorithm is made up of discrete steps, each of which consists of performing an elementary operation or inspecting an elementary condition unless it is the initial step or a final step;
3. the pattern of behaviour expressed by a classical algorithm is such that there is one possible step immediately following a step that consists of performing an operation;
4. the pattern of behaviour expressed by a classical algorithm is such that there is one possible step immediately following a step that consists of inspecting a condition for each outcome of the inspection;
5. the pattern of behaviour expressed by a classical algorithm is such that the initial step consists of inputting an input value of the problem concerned;
6. the pattern of behaviour expressed by a classical algorithm is such that, for each input value of the problem concerned for which a correct

output value exists, a final step is reached after a finite number of steps and that final step consists of outputting a correct output value for that input value;

7. the steps involved in the pattern of behaviour expressed by a classical algorithm are precisely and unambiguously defined and can be done exactly in a finite amount of time.

In the publications in which the viewpoints are expressed from which properties 1–7 are derived, it is usually mentioned at most in passing that a classical algorithm expresses a pattern of behaviour. Following [4], this point is central in this paper.

By property 3, a classical algorithm is a deterministic algorithm. I coin the term non-interactive algorithm for the notion of an algorithm characterized by the same properties as the notion of a classical algorithm except that the restriction to deterministic algorithms is dropped, i.e. with property 3 replaced by following property:

3′ the pattern of behaviour expressed by a classical algorithm is such that there is at least one possible step immediately following a step that consists of performing an operation.

In this paper, properties 1, 2, 3′, 4, 5, 6, and 7 are considered to give an intuitive characterization of the notion of a non-interactive algorithm and will be used as the starting point for the formalization of this notion.

The viewpoints on interactive computation that are expressed in computer science publications such as [3, 6, 13] are the basis of the following description of an interactive computational problem:

> An interactive computational problem is a problem where, given a possibly infinite sequence of values from a certain set of input values, a possibly infinite sequence of values from a certain set of output values that is in a certain causal relation to the given sequence of input values must be produced if it exists. Causal here means that, for each n, for each two sequences of input values with a common prefix of length n, for each sequence of output values with a prefix of length n that is related to one of those two sequences of input values, there exists a sequence of output values with the same prefix that is related to the other of those two sequences of input values. The sequences of input values are called the instances of the problem and a sequence of

output values that is in the certain relation to a given sequence of input values is called a solution for the instance concerned.

Classical computational problems can be considered the simplest interactive computational problems.

Classical algorithms are not adequate for solving interactive computational problems. From the viewpoints on interactive computation referred to above, it follows that:

- with the exception of properties 2, 3, and 6, the most important properties of a classical algorithm are also considered to belong to the most important properties of an interactive algorithm;
- the following generalizations of properties 2 and 6 are considered to belong to the most important properties of an interactive algorithm:

  2′. the pattern of behaviour expressed by an interactive algorithm is made up of discrete steps, each of which consists of performing an elementary operation or inspecting an elementary condition unless it is the initial step, a final step, an input step or an output step;

  6′. the pattern of behaviour expressed by an interactive algorithm is such that, for each finite sequence of input values of the problem concerned for which a correct sequence of output values exists, after this sequence has been inputted, a final step is reached after a finite number of steps and that final step consists of outputting the last output value of a correct sequence of output values;

- the following property is also considered to belong to the most important ones of an interactive algorithm:

  8. the pattern of behaviour expressed by an interactive algorithm is such that:
     a. there is one possible step immediately following an output step;
     b. an output step is immediately followed by an input step and an input step immediately follows an output step;
     c. an input step consists of inputting an input value of the problem concerned and an output step consists of outputting an output value that is correct for the sequence of input values inputted so far.

From the viewpoints on interactive computation referred to above, it is not clear whether property 3 should be considered to belong to the most important properties of an interactive algorithm. Therefore, in this paper, properties 1, $2'$, $3'$, 4, 5, $6'$, 7, and 8 are considered to give an intuitive characterization of the notion of an interactive algorithm and will be used as the starting point for the formalization of this notion.

4 Non-Interactive Algorithms

In this section, the notion of a non-interactive proto-algorithm is introduced first. The thought is that non-interactive algorithms are equivalence classes of non-interactive proto-algorithms under some equivalence relation. That is why three equivalence relations are defined next. Two of them give bounds between which an appropriate equivalence relation must lie. The third lies in between these two and is likely an appropriate equivalence relation.

4.1 Non-Interactive Proto-Algorithms

The notion of a non-interactive proto-algorithm will be defined in terms of three auxiliary notions. First, we define these auxiliary notions, starting with the notion of a non-interactive alphabet.

Definition 5 *A* non-interactive alphabet Σ *is a couple* (F, P)*, where:*

- F *is a countable set of* function symbols *of* Σ*;*
- P *is a countable set of* predicate symbols *of* Σ*;*
- F *and* P *are disjoint sets and* $\mathsf{ini}, \mathsf{fin} \in F$*.*

We write $\widehat{F}$ and $\widetilde{F}$, where F is the set of function symbols of a non-interactive alphabet, for the sets $F \setminus \{\mathsf{fin}\}$ and $F \setminus \{\mathsf{ini}, \mathsf{fin}\}$, respectively.

In the case of a non-interactive alphabet (F, P), the function symbols from $\widetilde{F}$ and predicate symbols from P refer to the operations and conditions, respectively, involved in the steps of which the pattern of behaviour expressed by a non-interactive algorithm is made up. The function symbols ini and fin refer to inputting an input value and outputting an output value, respectively.

We are now ready to define the notions of a non-interactive Σ-algorithm graph and a non-interactive Σ-interpretation. They concern the pattern of behaviour expressed by a non-interactive algorithm.

Definition 6 *Let $\Sigma = (F, P)$ be a non-interactive alphabet. Then a* non-interactive Σ-algorithm graph *G is a rooted labeled directed graph $(V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$ such that*

- $L_{\mathrm{v}} = F \cup P$*;*
- $L_{\mathrm{e}} = \{0, 1\}$*;*
- *for all $v \in V$:*
  - $\mathrm{indeg}(v) = 0$ *iff* $v = r$*;*
  - $l(v) = \mathsf{ini}$ *iff* $\mathrm{indeg}(v) = 0$*;*
  - $l(v) = \mathsf{fin}$ *iff* $\mathrm{outdeg}(v) = 0$*;*
  - *if $l(v) \in F$, then, for each $v' \in V$ such that $(v, v') \in E$, $l((v, v'))$ is undefined;*
  - *if $l(v) \in P$, then* $\mathrm{outdeg}(v) = 2$ *and, for the unique $v', v'' \in V$ such that $v' \neq v''$ and $(v, v'), (v, v'') \in E$, both $l((v, v'))$ and $l((v, v''))$ are defined and $l((v, v')) \neq l((v, v''))$;*
- *for all cycles $v_1 \ldots v_{n+1}$ in G, there exists a $v \in \{v_1, \ldots, v_n\}$ such that $l(v) \in F$.*

Non-interactive Σ-algorithm graphs are somewhat reminiscent of program schemes as defined, for example, in [14].

In the above definition, the condition on cycles in a non-interactive Σ-algorithm graph excludes infinitely many consecutive steps, each of which consists of inspecting a condition.

In the above definition, the conditions regarding the vertices and edges of a non-interactive Σ-algorithm graph correspond to the essential properties of a non-interactive algorithm described in Section 3 that concern its structure. Adding an interpretation of the symbols of the non-interactive alphabet Σ to a non-interactive Σ-algorithm graph yields something that has all of the essential properties of a non-interactive algorithm described in Section 3.

Definition 7 *Let $\Sigma = (F, P)$ be a non-interactive alphabet. Then a* non-interactive Σ-interpretation *$\mathcal{I}$ is a quadruple $(D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, where:*

- *D is a set, called the* algorithm domain *of $\mathcal{I}$;*
- *D_{i} is a finitely generated set, called the* input domain *of $\mathcal{I}$;*

- D_o *is a finitely generated set, called the* output domain *of* $\mathcal{I}$*;*
- I *is a total function from* $F \cup P$ *to the set of all total computable functions from* D_i *to* D*,* D *to* D_o*,* D *to* D *or* D *to* $\{0,1\}$ *such that:*
  - $I(\mathsf{ini})$ *is a function from* D_i *to* D*;*
  - $I(\mathsf{fin})$ *is a function from* D *to* D_o*;*
  - *for all* $f \in \widetilde{F}$*,* $I(f)$ *is a function from* D *to* D*;*
  - *for all* $p \in P$*,* $I(p)$ *is a function from* D *to* $\{0,1\}$*;*
- *there does not exist a* $D' \subset D$ *such that:*
  - *for all* $d_\mathrm{i} \in D_\mathrm{i}$*,* $I(\mathsf{ini})(d_\mathrm{i}) \in D'$*;*
  - *for all* $f \in \widetilde{F}$*, for all* $d \in D'$*,* $I(f)(d) \in D'$*.*

The finite generation condition on D_i and D_o and the minimality condition on D in the above definition are considered desirable. They guarantee that all elements of D_i, D_o, and D have a finite representation, which is generally expected of the values involved in the steps of an algorithm. However, these conditions have not yet been shown to be essential in establishing results.

The pattern of behavior expressed by a non-interactive algorithm can be fully represented by the combination of a non-interactive alphabet Σ, a non-interactive Σ-algorithm graph G, and a non-interactive Σ-interpretation $\mathcal{I}$. This brings us to defining the notion of a non-interactive proto-algorithm.

Definition 8 *A* non-interactive proto-algorithm *A is a triple* $(\Sigma, G, \mathcal{I})$*, where:*

- Σ *is a non-interactive alphabet, called the* alphabet *of* A*;*
- G *is a non-interactive* Σ*-algorithm graph, called the* algorithm graph *of* A*;*
- $\mathcal{I}$ *is a non-interactive* Σ*-interpretation, called the* interpretation *of* A*.*

A distinction can be made between deterministic non-interactive proto-algorithms and non-deterministic non-interactive proto-algorithms.

Definition 9 *Let* $A = (\Sigma, G, \mathcal{I})$ *be a non-interactive proto-algorithm, where* $\Sigma = (F, P)$ *and* $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$*. Then* A is deterministic *if, for all* $v \in V$ *with* $l(v) \in \widehat{F}$*,* $\mathrm{outdeg}(v) = 1$*, and* A is non-deterministic *if* A *is not deterministic.*

Henceforth, we assume a dummy value $\bot$ such that, for all non-interactive proto-algorithms $A = (\Sigma, G, \mathcal{I})$, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$, $\bot \notin V$, $\bot \notin D$, $\bot \notin D_\mathrm{i}$, and $\bot \notin D_\mathrm{o}$.

The intuition is that a non-interactive proto-algorithm is something that goes from one state to another.

Definition 10 *Let $A = (\Sigma, G, \mathcal{I})$ be a non-interactive proto-algorithm, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$. Then a* state *of A is a triple $(d_\mathrm{i}, (v, d), d_\mathrm{o}) \in D_{\mathrm{i}\bot} \times (V_\bot \times D_\bot) \times D_{\mathrm{o}\bot}$ such that:*

- *$v = \bot$ iff $d = \bot$;*
- *if $(v, d) = (\bot, \bot)$, then $d_\mathrm{i} = \bot$ iff $d_\mathrm{o} \neq \bot$;*
- *if $(v, d) \neq (\bot, \bot)$, then $d_\mathrm{i} = \bot$ and $d_\mathrm{o} = \bot$.*

A state s of A is

- *an* initial state *of A if $s \in D_\mathrm{i} \times \{(\bot, \bot)\} \times \{\bot\}$;*
- *a* final state *of A if $s \in \{\bot\} \times \{(\bot, \bot)\} \times D_\mathrm{o}$;*
- *an* internal state *of A if $s \in \{\bot\} \times (V \times D) \times \{\bot\}$.*

Henceforth, we write $\mathcal{S}_A$, where A is a non-interactive proto-algorithm, for the set of all states of A. We also write $\mathcal{S}_A^{\mathrm{ini}}$, $\mathcal{S}_A^{\mathrm{fin}}$, and $\mathcal{S}_A^{\mathrm{int}}$ for the set of all initial states of $\mathcal{S}_A$, the set of all final states of $\mathcal{S}_A$, and the set of all internal states of $\mathcal{S}_A$, respectively. Moreover, we write $\bot_\mathrm{c}$ for the tuple $(\bot, \bot)$.

An internal state $(\bot, (v, d), \bot)$ of a non-interactive proto-algorithm A can be largely explained from the point of view of state machines:

- v is the control state of A;
- d is the data state of A.

Suppose that $A = (\Sigma, G, \mathcal{I})$ is a non-interactive proto-algorithm, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$. A goes from one state to the next state by making a step, it starts in an initial state, and if it does not keep making steps forever, it stops in a final state. The following is an informal explanation of how the state that A is in determines what the possible steps to a next state consists of and to what next states they lead:

- if A is in initial state $(d_\mathrm{i}, \bot_\mathrm{c}, \bot)$, then a step to a next state is possible that consists of applying function $I(\mathsf{ini})$ to d_i and that leads to one of the internal states $(\bot, (v', d'), \bot)$ such that $(r, v') \in E$ and $I(\mathsf{ini})(d_\mathrm{i}) = d'$;
- if A is in internal state $(\bot, (v, d), \bot)$ and $l(v) \in \widetilde{F}$, then a step to a next state is possible that consists of applying function $I(l(v))$ to d and that leads to one of the internal states $(\bot, (v', d'), \bot)$ such that $(v, v') \in E$ and $I(l(v))(d) = d'$;
- if A is in internal state $(\bot, (v, d), \bot)$ and $l(v) \in P$, then a step to a next state is possible that consists of applying function $I(l(v))$ to d and that leads to the unique internal state $(\bot, (v', d), \bot)$ such that $(v, v') \in E$ and $I(l(v))(d) = l((v, v'))$;
- if A is in internal state $(\bot, (v, d), \bot)$ and $l(v) = \mathsf{fin}$, then a step to a next state is possible that consists of applying function $I(\mathsf{fin})$ to d and that leads to the unique final state $(\bot, \bot_\mathrm{c}, d_\mathrm{o})$ such that $I(\mathsf{fin})(v) = d_\mathrm{o}$.

This informal explanation of how the state that A is in determines what the possible next states are, is formalized by the algorithmic step function δ^a_A defined in Section 4.2.

Because non-interactive proto-algorithms are considered too concrete to be called non-interactive algorithms, the term non-interactive proto-algorithm has been chosen instead of the term non-interactive algorithm. For example, from a mathematical point of view, it is natural to consider the behavioral patterns expressed by isomorphic non-interactive proto-algorithms to be the same.

It should be intuitively clear what isomorphism of non-interactive proto-algorithms is. For the sake of completeness, here is a precise definition.

Definition 11 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be non-interactive proto-algorithms, where $\Sigma = (F, P)$, $\Sigma' = (F', P')$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, $G' = (V', E', L_\mathrm{v}', L_\mathrm{e}', l', r')$, $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$, and $\mathcal{I}' = (D', D'_\mathrm{i}, D'_\mathrm{o}, I')$. Then A and A' are* isomorphic*, written $A \cong A'$, if there exist bijections $\beta_\mathrm{f} : F \to F'$, $\beta_\mathrm{p} : P \to P'$, $\beta_\mathrm{v} : V \to V'$, $\beta_\mathrm{d} : D \to D'$, $\beta_{\mathrm{d_i}} : D_\mathrm{i} \to D'_\mathrm{i}$, $\beta_{\mathrm{d_o}} : D_\mathrm{o} \to D'_\mathrm{o}$, and $\beta_\mathrm{b} : \{0,1\} \to \{0,1\}$ such that:*

- *for all $v, v' \in V$, $(v, v') \in E$ iff $(\beta_\mathrm{v}(v), \beta_\mathrm{v}(v')) \in E'$;*
- *for all $v \in V$ with $l(v) \in F$, $\beta_\mathrm{f}(l(v)) = l'(\beta_\mathrm{v}(v))$;*
- *for all $v \in V$ with $l(v) \in P$, $\beta_\mathrm{p}(l(v)) = l'(\beta_\mathrm{v}(v))$;*

- *for all* $(v, v') \in E$ *with* $l((v, v'))$ *defined,* $\beta_{\mathrm{b}}(l((v, v'))) = l'((\beta_{\mathrm{v}}(v), \beta_{\mathrm{v}}(v')))$*;*
- $\beta_{\mathrm{f}}(\mathsf{ini}) = \mathsf{ini}$ *and* $\beta_{\mathrm{f}}(\mathsf{fin}) = \mathsf{fin}$*;*
- *for all* $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $\beta_{\mathrm{d}}(I(\mathsf{ini})(d_{\mathrm{i}})) = I'(\mathsf{ini})(\beta_{\mathrm{d_i}}(d_{\mathrm{i}}))$*;*
- *for all* $d \in D$, $\beta_{\mathrm{d_o}}(I(\mathsf{fin})(d)) = I'(\mathsf{fin})(\beta_{\mathrm{d}}(d))$*;*
- *for all* $d \in D$ *and* $f \in \widetilde{F}$, $\beta_{\mathrm{d}}(I(f)(d)) = I'(\beta_{\mathrm{f}}(f))(\beta_{\mathrm{d}}(d))$*;*
- *for all* $d \in D$ *and* $p \in P$, $\beta_{\mathrm{b}}(I(p)(d)) = I'(\beta_{\mathrm{p}}(p))(\beta_{\mathrm{d}}(d))$*.*

Non-interactive proto-algorithms may also be considered too concrete in a way not covered by isomorphism of non-interactive proto-algorithms. This issue is addressed in Section 4.3 and leads there to the introduction of two other equivalence relations.

A non-interactive proto-algorithm could also be defined as a quadruple $(D, D_{\mathrm{i}}, D_{\mathrm{o}}, \overline{G})$ where $\overline{G}$ is a graph that differs from a non-interactive Σ-algorithm graph in that its vertex labels are computable functions from D_{i} to D, D to D_{o}, D to D or D to $\{0, 1\}$ instead of function and predicate symbols from Σ. I consider the definition of a non-interactive proto-algorithm given earlier more insightful because it isolates as much as possible the operations to be performed and the conditions to be inspected from the structure of a non-interactive proto-algorithm.

4.2 On Steps, Runs and What is Computed

In Section 4.1, the intuition was given that a non-interactive proto-algorithm A is something that goes through states. It was informally explained how the state that it is in determines what the possible steps to a next state consists of and to what next states they lead. The algorithmic step function δ^{a}_A that is defined below formalizes this. The computational step function δ^{c}_A that is also defined below is like the algorithmic step function δ^{a}_A, but conceals the steps that consist of inspecting conditions.

Definition 12 *Let* $A = (\Sigma, G, \mathcal{I})$ *be a non-interactive proto-algorithm, where* $\Sigma = (F, P)$, $G = (V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$, *and* $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$*. Then the* algorithmic step function δ^{a}_A induced by A *is the smallest total function from* $\mathcal{S}_A$ *to* $\mathcal{P}(\mathcal{S}_A) \setminus \emptyset$ *such that for all* $d, d' \in D$, $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $d_{\mathrm{o}} \in D_{\mathrm{o}}$, *and* $v' \in V$*:*[1]

[1] Below, we write $S \ni s$ instead of $s \in S$.

$$\begin{aligned}
\delta^{\mathrm{a}}_A((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot)) &\ni (\bot, (v', d'), \bot) \\
&\quad \textit{if } l(r) = \mathsf{ini},\ (r, v') \in E,\ \textit{and } I(\mathsf{ini})(d_{\mathrm{i}}) = d', \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (\bot, (v', d'), \bot) \\
&\quad \textit{if } l(v) \in \widetilde{F},\ (v, v') \in E,\ \textit{and } I(l(v))(d) = d', \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (\bot, (v', d), \bot) \\
&\quad \textit{if } l(v) \in P,\ (v, v') \in E,\ \textit{and } I(l(v))(d) = l((v, v')), \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}}) \\
&\quad \textit{if } l(v) = \mathsf{fin} \textit{ and } I(\mathsf{fin})(d) = d_{\mathrm{o}}, \\
\delta^{\mathrm{a}}_A((\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})
\end{aligned}$$

and the computational step function δ^{c}_A induced by A *is the smallest total function from* $\mathcal{S}_A$ *to* $\mathcal{P}(\mathcal{S}_A) \setminus \emptyset$ *such that, for all* $d, d' \in D$, $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $d_{\mathrm{o}} \in D_{\mathrm{o}}$, $v' \in V$, *and* $s \in \mathcal{S}_A$*:*

$$\begin{aligned}
\delta^{\mathrm{c}}_A((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot)) &\ni (\bot, (v', d'), \bot) \\
&\quad \textit{if } l(r) = \mathsf{ini},\ (r, v') \in E,\ \textit{and } I(\mathsf{ini})(d_{\mathrm{i}}) = d', \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni (\bot, (v', d'), \bot) \\
&\quad \textit{if } l(v) \in \widetilde{F},\ (v, v') \in E,\ \textit{and } I(l(v))(d) = d', \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni s \\
&\quad \textit{if } l(v) \in P,\ (v, v') \in E,\ I(l(v))(d) = l((v, v')),\ \textit{and} \\
&\qquad \delta^{\mathrm{c}}_A((\bot, (v', d), \bot)) \ni s, \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}}) \\
&\quad \textit{if } l(v) = \mathsf{fin} \textit{ and } I(\mathsf{fin})(d) = d_{\mathrm{o}}, \\
\delta^{\mathrm{c}}_A((\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}}).
\end{aligned}$$

Below, we define what the possible runs of a non-interactive proto-algorithm on an input value are and what is computed by a non-interactive proto-algorithm. To this end, we first give some auxiliary definitions.

The functions srs^{a}_A and srs^{c}_A defined below give the sets of sequences of states that A goes through from a given state according to the algorithmic step function and computational step function, respectively.

Definition 13 *Let* $A = (\Sigma, G, \mathcal{I})$ *be a non-interactive proto-algorithm. Then the* algorithmic semi-run set function srs^{a}_A induced by A *and the* computational semi-run set function srs^{c}_A induced by A *are the total functions from* $\mathcal{S}_A$ *to* $\mathcal{P}({\mathcal{S}_A}^{\infty})$ *such that for all* $s \in \mathcal{S}_A$*:*

$$\begin{aligned}
srs^{\mathrm{a}}_A(s) &= \{\langle s \rangle \frown \sigma \mid (\exists s' \in \delta^{\mathrm{a}}_A(s))\ \sigma \in srs^{\mathrm{a}}_A(s')\} && \textit{if } s \notin \mathcal{S}^{\mathrm{fin}}_A, \\
srs^{\mathrm{a}}_A(s) &= \{\langle s \rangle\} && \textit{if } s \in \mathcal{S}^{\mathrm{fin}}_A.
\end{aligned}$$

and

$$srs^{\mathrm{c}}_A(s) = \{\langle s\rangle \frown \sigma \mid (\exists s' \in \delta^{\mathrm{c}}_A(s))\ \sigma \in srs^{\mathrm{c}}_A(s')\} \text{ if } s \notin \mathcal{S}^{\mathrm{fin}}_A,$$
$$srs^{\mathrm{c}}_A(s) = \{\langle s\rangle\} \text{ if } s \in \mathcal{S}^{\mathrm{fin}}_A.$$

A $\sigma \in {\mathcal{S}_A}^{\infty}$ *is called an* algorithmic semi-run *if there exists an* $s \in \mathcal{S}_A$ *such that* $\sigma \in srs^{\mathrm{a}}_A(s)$.

Henceforth, we write $\mathcal{SR}^{\mathrm{a}}_A$, where A is a non-interactive proto-algorithm, for the set of all algorithmic semi-runs.

When defining what is computed by a non-interactive proto-algorithm, a distinction must be made between convergent algorithmic semi-runs and divergent semi-runs.

Definition 14 *Let* $A = (\Sigma, G, \mathcal{I})$ *be a non-interactive proto-algorithm, and let* $\sigma \in \mathcal{SR}^{\mathrm{a}}_A$. *Then* σ is divergent *if there exists a suffix* σ' *of* σ *such that* $\sigma' \in {\mathcal{S}^{\mathrm{int}}_A}^{\infty}$, *and* σ is convergent *if* σ *is not divergent.*

The function ov_A defined below gives the output value eventually produced by a given convergent algorithmic semi-run.

Definition 15 *Let* $A = (\Sigma, G, \mathcal{I})$ *be a non-interactive proto-algorithm. Then the* output value extraction function ov_A for A *is the total function from* $\{\sigma \in \mathcal{SR}^{\mathrm{a}}_A \mid \sigma \text{ is convergent}\}$ *to* D_{o} *such that for all* $(d_{\mathrm{i}}, c, d_{\mathrm{o}}) \in \mathcal{S}_A$ *and* $\sigma \in \mathcal{S}^{\infty}_A \cup \{\langle\rangle\}$ *such that* $\langle(d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma \in \mathcal{SR}^{\mathrm{a}}_A$ *and* $\langle(d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma$ *is convergent:*

$$ov_A(\langle(d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = d_{\mathrm{o}} \quad \text{if } d_{\mathrm{o}} \neq \bot,$$
$$ov_A(\langle(d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = ov_A(\sigma) \text{ if } d_{\mathrm{o}} = \bot.$$

A run of a non-interactive proto-algorithm is a semi-run of the non-interactive proto-algorithm concerned that starts in an initial state. The following definition concerns what the possible runs of a non-interactive proto-algorithm on an input value are. As with steps, a distinction is made between algorithmic runs and computational runs.

Definition 16 *Let* $A = (\Sigma, G, \mathcal{I})$ *be a non-interactive proto-algorithm, where* $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, *and let* $d_{\mathrm{i}} \in D_{\mathrm{i}}$. *Then the* algorithmic run set of A on d_{i}, *written* $\rho^{\mathrm{a}}_A(d_{\mathrm{i}})$, *is* $srs^{\mathrm{a}}_A((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot))$ *and the* computational run set of A on d_{i}, *written* $\rho^{\mathrm{c}}_A(d_{\mathrm{i}})$, *is* $srs^{\mathrm{c}}_A((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot))$.

The following definition concerns what is computed by a non-interactive proto-algorithm.

Definition 17 *Let $A = (\Sigma, G, \mathcal{I})$ be a non-interactive proto-algorithm, where $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$. Then the* relation $\widehat{A}$ computed by A *is the relation from D_{i} to D_{o} such that for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$ and $d_{\mathrm{o}} \in D_{\mathrm{o}}$:*

$(d_{\mathrm{i}}, d_{\mathrm{o}}) \in \widehat{A}$ *iff there exists a convergent $\sigma \in \rho^{\mathrm{a}}_A(d_{\mathrm{i}})$ such that $d_{\mathrm{o}} = ov_A(\sigma)$.*

If A is a deterministic non-interactive proto-algorithm, then the relation $\widehat{A}$ computed by A is functional, i.e. $\widehat{A}$ is (the graph of) a partial function.

4.3 Algorithmic and Computational Equivalence

If a non-interactive proto-algorithm A' can mimic a non-interactive proto-algorithm A step-by-step, then we say that A is algorithmically simulated by A'. If the steps that consist of inspecting conditions are ignored, then we say that A is computationally simulated by A'. Algorithmic and computational simulation can be formally defined using the step functions defined in Section 4.2.

Definition 18 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be non-interactive proto-algorithms. Then an* algorithmic simulation of A by A' *is a set $R \subseteq \mathcal{S}_A \times \mathcal{S}_{A'}$ such that:*

- *for all $s \in \mathcal{S}_A$ and $s' \in \mathcal{S}_{A'}$:*
  - *if $s \in \mathcal{S}^{\mathrm{ini}}_A$, then there exists an $s' \in \mathcal{S}^{\mathrm{ini}}_{A'}$ such that $(s, s') \in R$;*
  - *if $s' \in \mathcal{S}^{\mathrm{fin}}_{A'}$, then there exists an $s \in \mathcal{S}^{\mathrm{fin}}_A$ such that $(s, s') \in R$;*
  - *if $(s, s') \in R$ and $t \in \delta^{\mathrm{a}}_A(s)$, then there exists a $t' \in \delta^{\mathrm{a}}_{A'}(s')$ such that $(t, t') \in R$;*
- *for all $(s, s') \in R$:*
  - *$s \in \mathcal{S}^{\mathrm{ini}}_A$ iff $s' = \mathcal{S}^{\mathrm{ini}}_{A'}$;*
  - *$s \in \mathcal{S}^{\mathrm{fin}}_A$ iff $s' = \mathcal{S}^{\mathrm{fin}}_{A'}$*

and a computational simulation of A by A' *is a set $R \subseteq \mathcal{S}_A \times \mathcal{S}_{A'}$ such that:*

- *for all $s \in \mathcal{S}_A$ and $s' \in \mathcal{S}_{A'}$:*
  - *if $s \in \mathcal{S}^{\mathrm{ini}}_A$, then there exists an $s' \in \mathcal{S}^{\mathrm{ini}}_{A'}$ such that $(s, s') \in R$;*
  - *if $s' \in \mathcal{S}^{\mathrm{fin}}_{A'}$, then there exists an $s \in \mathcal{S}^{\mathrm{fin}}_A$ such that $(s, s') \in R$;*

  - *if* $(s, s') \in R$ *and* $t \in \delta^{\mathrm{c}}_A(s)$*, then there exists a* $t' \in \delta^{\mathrm{c}}_{A'}(s')$ *such that* $(t, t') \in R$*;*

- *for all* $(s, s') \in R$*:*

  - $s \in \mathcal{S}^{\mathrm{ini}}_A$ *iff* $s' = \mathcal{S}^{\mathrm{ini}}_{A'}$*;*
  - $s \in \mathcal{S}^{\mathrm{fin}}_A$ *iff* $s' = \mathcal{S}^{\mathrm{fin}}_{A'}$*.*

A is algorithmically simulated by A', *written* $A \sqsubseteq_{\mathrm{a}} A'$, *if there exists an algorithmic simulation* R *of* A *by* A'.
A is computationally simulated by A', *written* $A \sqsubseteq_{\mathrm{c}} A'$, *if there exists a computational simulation* R *of* A *by* A'.
A is algorithmically equivalent to A', *written* $A \equiv_{\mathrm{a}} A'$, *if there exist an algorithmic simulation* R *of* A *by* A' *and an algorithmic simulation* R' *of* A' *by* A *such that* $R' = R^{-1}$.
A is computationally equivalent to A', *written* $A \equiv_{\mathrm{c}} A'$, *if there exist a computational simulation* R *of* A *by* A' *and a computational simulation* R' *of* A' *by* A *such that* $R' = R^{-1}$.

The conditions imposed on an algorithmic or computational simulation R of a non-interactive proto-algorithm A by a non-interactive proto-algorithm A' include, in addition to the usual transfer conditions, also conditions that guarantee that a state of A is only related by R to a state of A' of the same kind (initial, final or internal). Lemma 19 given below, used in the proof of the theorem that follows it, would not hold without the additional conditions.

There may be states of a non-interactive proto-algorithm in which there is a choice from multiple possible steps to a next state. The condition $R' = R^{-1}$ imposed on simulations R and R' witnessing (algorithmic or computational) equivalence of non-interactive proto-algorithms A and A' guarantees that A and A' have the same choice structure.

Lemma 19 *Let* $A = (\Sigma, G, \mathcal{I})$ *and* $A' = (\Sigma', G', \mathcal{I}')$ *be non-interactive proto-algorithms, where* $\Sigma = (F, P)$, $\Sigma' = (F', P')$, $G = (V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$, $G' = (V', E', L_{\mathrm{v}}{}', L_{\mathrm{e}}{}', l', r')$, $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, *and* $\mathcal{I}' = (D', D'_{\mathrm{i}}, D'_{\mathrm{o}}, I')$, *and let* $R \subseteq \mathcal{S}_A \times \mathcal{S}_{A'}$ *and* $\gamma_{\mathrm{i}} : D_{\mathrm{i}} \to D'_{\mathrm{i}}$ *be such that, for all* $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot), (\gamma_{\mathrm{i}}(d_{\mathrm{i}}), \bot_{\mathrm{c}}, \bot)) \in R$. *Then* R *is an algorithmic simulation of* A *by* A' *only if, for all* $d_{\mathrm{i}} \in D_{\mathrm{i}}$, *for all* $\sigma \in \rho^{\mathrm{a}}_A(d_{\mathrm{i}})$, *there exists a* $\sigma' \in \rho^{\mathrm{a}}_{A'}(\gamma_{\mathrm{i}}(d_{\mathrm{i}}))$ *such that, for all* $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$.

Proof: Let R be an algorithmic simulation of A by A' and $\gamma_{\mathrm{i}} : D_{\mathrm{i}} \to D'_{\mathrm{i}}$ be such that, for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot), (\gamma_{\mathrm{i}}(d_{\mathrm{i}}), \bot_{\mathrm{c}}, \bot)) \in R$, let $d_{\mathrm{i}} \in D_{\mathrm{i}}$, and let $\sigma \in \rho^{\mathrm{a}}_A(d_{\mathrm{i}})$. Then we can easily construct a $\sigma' \in \rho^{\mathrm{a}}_{A'}(\gamma_{\mathrm{i}}(d_{\mathrm{i}}))$ such that, for all $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$, using the conditions imposed on algorithmic simulations. □

The following theorem tells us that, if a non-interactive proto-algorithm A is algorithmically simulated by a non-interactive proto-algorithm A', then (a) the relation computed by A' models the relation computed by A (in the sense of e.g. [7]) and (b) for each convergent algorithmic run of A, the simulation results in a convergent algorithmic run of A' consisting of the same number of algorithmic steps.

Theorem 20 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be non-interactive proto-algorithms, where $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, and $\mathcal{I}' = (D', D'_{\mathrm{i}}, D'_{\mathrm{o}}, I')$. Then $A \sqsubseteq_{\mathrm{a}} A'$ only if there exist total functions $\gamma_{\mathrm{i}} : D_{\mathrm{i}} \to D'_{\mathrm{i}}$ and $\gamma_{\mathrm{o}} : D'_{\mathrm{o}} \to D_{\mathrm{o}}$ such that:*

(1) for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$ and $d_{\mathrm{o}} \in D_{\mathrm{o}}$, $(d_{\mathrm{i}}, d_{\mathrm{o}}) \in \widehat{A}$ only if there exists a $d'_{\mathrm{o}} \in D'_{\mathrm{o}}$ such that $(\gamma_{\mathrm{i}}(d_{\mathrm{i}}), d'_{\mathrm{o}}) \in \widehat{A'}$ and $\gamma_{\mathrm{o}}(d'_{\mathrm{o}}) = d_{\mathrm{o}}$;

(2) for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$, for all convergent $\sigma \in \rho^{\mathrm{a}}_A(d_{\mathrm{i}})$, there exists a convergent $\sigma' \in \rho^{\mathrm{a}}_{A'}(\gamma_{\mathrm{i}}(d_{\mathrm{i}}))$ with $\gamma_{\mathrm{o}}(ov_{A'}(\sigma')) = ov_A(\sigma)$ such that $|\sigma| = |\sigma'|$.

Proof: Assume that $A \sqsubseteq_{\mathrm{a}} A'$. Let R be an algorithmic simulation of A by A', let γ_{i} be a function from D_{i} to D'_{i} such that, for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot), (\gamma_{\mathrm{i}}(d_{\mathrm{i}}), \bot_{\mathrm{c}}, \bot)) \in R$, and let γ_{o} be a function from D'_{o} to D_{o} such that, for all $d'_{\mathrm{o}} \in D'_{\mathrm{o}}$, $((\bot, \bot_{\mathrm{c}}, \gamma_{\mathrm{o}}(d'_{\mathrm{o}})), (\bot, \bot_{\mathrm{c}}, d'_{\mathrm{o}})) \in R$. The algorithmic simulation R exists by the definition of $\sqsubseteq_{\mathrm{a}}$, and the functions γ_{i} and γ_{o} exist by the definition of an algorithmic simulation. By Lemma 19, for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$, for all $\sigma \in \rho^{\mathrm{a}}_A(d_{\mathrm{i}})$, there exists a $\sigma' \in \rho^{\mathrm{a}}_{A'}(\gamma_{\mathrm{i}}(d_{\mathrm{i}}))$ such that, for all $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$.

Let $d_{\mathrm{i}} \in D_{\mathrm{i}}$, and let $\sigma \in \rho^{\mathrm{a}}_A(d_{\mathrm{i}})$ and $\sigma' \in \rho^{\mathrm{a}}_{A'}(\gamma_{\mathrm{i}}(d_{\mathrm{i}}))$ be such that, for all $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$. Then from the definition of an algorithmic simulation, it immediately follows that, for all $n \in \mathbb{N}^+$:

(a) $\sigma[n] \notin \mathcal{S}^{\mathrm{int}}_A$ only if $\sigma'[n] \notin \mathcal{S}^{\mathrm{int}}_{A'}$;

(b) for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$ and $d_{\mathrm{o}} \in D_{\mathrm{o}}$:

- $\sigma[n] = (d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot)$ only if $\sigma'[n] = (\gamma_{\mathrm{i}}(d_{\mathrm{i}}), \bot_{\mathrm{c}}, \bot)$;

- $\sigma[n] = (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})$ only if there exists a $d'_{\mathrm{o}} \in D'_{\mathrm{o}}$ such that $\sigma'[n] = (\bot, \bot_{\mathrm{c}}, d'_{\mathrm{o}})$ and $\gamma_{\mathrm{o}}(d'_{\mathrm{o}}) = d_{\mathrm{o}}$.

By the definition of the relation computed by a non-interactive proto-algorithm, both (1) and (2) follow immediately from (a) and (b). □

It is easy to see that Theorem 20 goes through as far as (1) is concerned if algorithmic simulation is replaced by computational simulation. However, (2) does not go through if algorithmic simulation is replaced by computational simulation.

The following theorem tells us how isomorphism, algorithmic equivalence, and computational equivalence are related.

Theorem 21 *Let A and A' be non-interactive proto-algorithms. Then:*

(1) $A \cong A'$ *only if* $A \equiv_{\mathrm{a}} A'$ *(2)* $A \equiv_{\mathrm{a}} A'$ *only if* $A \equiv_{\mathrm{c}} A'$.

Proof: Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be non-interactive proto-algorithms, where $\Sigma = (F, P)$, $\Sigma' = (F', P')$, $G = (V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$, $G' = (V', E', L_{\mathrm{v}}{}', L_{\mathrm{e}}{}', l', r')$, $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, and $\mathcal{I}' = (D', D'_{\mathrm{i}}, D'_{\mathrm{o}}, I')$.

Part 1. Assume that $A \cong A'$. Let β_{v}, β_{d}, $\beta_{\mathrm{d_i}}$, and $\beta_{\mathrm{d_o}}$ be bijections as in the definition of $\cong$, let β^*_{v}, β^*_{d}, $\beta^*_{\mathrm{d_i}}$, and $\beta^*_{\mathrm{d_o}}$ be the extensions of β_{v}, β_{d}, $\beta_{\mathrm{d_i}}$, and $\beta_{\mathrm{d_o}}$, respectively, with the dummy value $\bot$ such that $\beta^*_{\mathrm{v}}(\bot) = \bot$, $\beta^*_{\mathrm{d}}(\bot) = \bot$, $\beta^*_{\mathrm{d_i}}(\bot) = \bot$, and $\beta^*_{\mathrm{d_o}}(\bot) = \bot$, and let β be the bijection from $\mathcal{S}_A$ to $\mathcal{S}_{A'}$ defined by $\beta(d_{\mathrm{i}}, (v, d), d_{\mathrm{o}}) = (\beta^*_{\mathrm{d_i}}(d_{\mathrm{i}}), (\beta^*_{\mathrm{v}}(v), \beta^*_{\mathrm{d}}(d)), \beta^*_{\mathrm{d_o}}(d_{\mathrm{o}}))$. Moreover, let $R = \{(s, \beta(s)) \mid s \in \mathcal{S}_A\}$ and $R' = \{(s, \beta^{-1}(s)) \mid s \in \mathcal{S}_{A'}\}$. Then R is an algorithmic simulation of A by A' and R' is an algorithmic simulation of A' by A. This is easily proved by showing that the conditions from the definition of an algorithmic simulation are satisfied for all $(s, s') \in R$ and for all $(s, s') \in R'$, respectively. Moreover, it follows immediately from the definitions of R and R' that $R' = R^{-1}$. Hence, $A \equiv_{\mathrm{a}} A'$.

Part 2. Assume that $A \equiv_{\mathrm{a}} A'$. Let R be an algorithmic simulation of A by A' such that R^{-1} is an algorithmic simulation of A' by A. Then R is also a computational simulation of A by A' and R^{-1} is also an computational simulation of A' by A. This is easily proved by showing that the conditions from the definition of a computational simulation are satisfied for all $(s, s') \in R$ and for all $(s, s') \in R'$, respectively. Hence, $A \equiv_{\mathrm{c}} A'$. □

The opposite implications do not hold in general. That is, there exist non-interactive proto-algorithms A and A' for which it does not hold that $A \cong A'$ if $A \equiv_{\mathrm{a}} A'$ and there exist non-interactive proto-algorithms A and A'

for which it does not hold that $A \equiv_{\mathrm{a}} A'$ if $A \equiv_{\mathrm{c}} A'$. In both cases, the construction of a general illustrating example is described in [11].

The definition of algorithmic equivalence suggests that it is reasonable to consider the patterns of behaviour expressed by algorithmically equivalent non-interactive proto-algorithms the same. This suggests in turn that non-interactive algorithms can be considered equivalence classes of non-interactive proto-algorithms under algorithmic equivalence. The definition of computational equivalence does not suggest that it is reasonable to consider the patterns of behaviour expressed by computationally equivalent non-interactive proto-algorithms the same because steps that consist of inspecting a condition are treated as if they do not belong to the patterns of behaviour. The relevance of the computational equivalence relation is that any equivalence relation that captures the sameness of the patterns of behaviour expressed by non-interactive proto-algorithms to a higher degree than the algorithmic equivalence relation must be finer than the computational equivalence relation.

5 Interactive Algorithms

In this section, the notion of an interactive proto-algorithm is introduced first. As with non-interactive algorithms, the thought is that interactive algorithms are equivalence classes of interactive proto-algorithms under some equivalence relation. Again, three equivalence relations are defined next, two of which give bounds between which an appropriate equivalence relation must lie and one of which is likely an appropriate equivalence relation.

5.1 Interactive Proto-Algorithms

Like the notion of a non-interactive proto-algorithm, the notion of an interactive proto-algorithm will be defined in terms of three auxiliary notions. First, we define these auxiliary notions, starting with the notion of an interactive alphabet. Whereas there are two special function symbols in a non-interactive alphabet, there are four special function symbols in an interactive alphabet.

Definition 22 *An* interactive alphabet Σ *is a couple* (F, P)*, where:*

- F *is a countable set of* function symbols *of* Σ*;*
- P *is a countable set of* predicate symbols *of* Σ*;*

- *F and P are disjoint sets and* $\mathsf{ini}, \mathsf{fin}, \mathsf{in}, \mathsf{out} \in F$.

We write $\widehat{F}$ and $\widetilde{F}$, where F is the set of function symbols of an interactive alphabet, for the sets $F \setminus \{\mathsf{fin}\}$ and $F \setminus \{\mathsf{ini}, \mathsf{fin}, \mathsf{in}, \mathsf{out}\}$, respectively.

In the case of an interactive alphabet (F, P), the function symbols from $\widetilde{F}$ and predicate symbols from P refer to the operations and conditions, respectively, involved in the steps of which the pattern of behaviour expressed by an interactive algorithm is made up. The function symbols ini and fin refer to inputting a first input value and outputting a last output value, respectively, and the function symbols in and out refer to inputting a non-first input value and outputting a non-last output value, respectively.

We are now ready to define the notions of an interactive Σ-algorithm graph and an interactive Σ-interpretation. They concern the pattern of behaviour expressed by an interactive algorithm.

The definition of the notion of an interactive Σ-algorithm graph differs from the definition of the notion of a non-interactive Σ-algorithm graph only by the addition of two conditions concerning the vertex label out.

Definition 23 *Let $\Sigma = (F, P)$ be an interactive alphabet. Then an* interactive Σ-algorithm graph *G is a rooted labeled directed graph $(V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$ such that*

- $L_{\mathrm{v}} = F \cup P$*;*
- $L_{\mathrm{e}} = \{0, 1\}$*;*
- *for all $v \in V$:*
  - $\mathrm{indeg}(v) = 0$ *iff* $v = r$*;*
  - $l(v) = \mathsf{ini}$ *iff* $\mathrm{indeg}(v) = 0$*;*
  - $l(v) = \mathsf{fin}$ *iff* $\mathrm{outdeg}(v) = 0$*;*
  - $l(v) = \mathsf{out}$ *only if* $\mathrm{outdeg}(v) = 1$*;*
  - *if $l(v) \in F$, then, for each $v' \in V$ such that $(v, v') \in E$, $l((v, v'))$ is undefined;*
  - *if $l(v) \in P$, then* $\mathrm{outdeg}(v) = 2$ *and, for the unique $v', v'' \in V$ such that $v' \neq v''$ and $(v, v'), (v, v'') \in E$, both $l((v, v'))$ and $l((v, v''))$ are defined and $l((v, v')) \neq l((v, v''))$;*
- *for all $(v, v') \in E$, $l(v) = \mathsf{out}$ iff $l(v') = \mathsf{in}$;*

- *for all cycles* $v_1 \ldots v_{n+1}$ *in* G*, there exists a* $v \in \{v_1, \ldots, v_n\}$ *such that* $l(v) \in F$*.*

In the above definition, the conditions regarding the vertices and edges of an interactive Σ-algorithm graph correspond to the essential properties of an interactive algorithm mentioned in Section 3 that concern its structure. Adding an interpretation of the symbols of the interactive alphabet Σ to an interactive Σ-algorithm graph yields something that has all of the mentioned essential properties of an interactive algorithm described in Section 3.

The definition of the notion of an interactive Σ-interpretation differs from the definition of the notion of a non-interactive Σ-interpretation only by the addition of interpretation conditions of the special function symbols in and out and the adaptation of the minimality condition.

Definition 24 *Let* $\Sigma = (F, P)$ *be an interactive alphabet. Then a* interactive Σ-interpretation $\mathcal{I}$ *is a quadruple* $(D, D_\mathrm{i}, D_\mathrm{o}, I)$*, where:*

- D *is a set, called the* algorithm domain *of* $\mathcal{I}$*;*
- D_i *is a finitely generated set, called the* input domain *of* $\mathcal{I}$*;*
- D_o *is a finitely generated set, called the* output domain *of* $\mathcal{I}$*;*
- I *is a total function from* $F \cup P$ *to the set of all total computable functions from* D_i *to* D*,* D *to* D_o*,* $D \times D_\mathrm{i}$ *to* D*,* D *to* D *or* D *to* $\{0,1\}$ *such that:*
  - $I(\mathsf{ini})$ *is a function from* D_i *to* D*;*
  - $I(\mathsf{fin})$ *is a function from* D *to* D_o*;*
  - $I(\mathsf{in})$ *is a function from* $D \times D_\mathrm{i}$ *to* D*;*
  - $I(\mathsf{out})$ *is a function from* D *to* D_o*;*
  - *for all* $f \in \widetilde{F}$*,* $I(f)$ *is a function from* D *to* D*;*
  - *for all* $p \in P$*,* $I(p)$ *is a function from* D *to* $\{0,1\}$*;*
- *there does not exist a* $D' \subset D$ *such that:*
  - *for all* $d_\mathrm{i} \in D_\mathrm{i}$*,* $I(\mathsf{ini})(d_\mathrm{i}) \in D'$*;*
  - *for all* $d_\mathrm{i} \in D_\mathrm{i}$*, for all* $d \in D'$*,* $I(\mathsf{in})(d, d_\mathrm{i}) \in D'$*;*
  - *for all* $f \in \widetilde{F}$*, for all* $d \in D'$*,* $I(f)(d) \in D'$*.*

As with non-interactive interpretations, the finite generation condition on D_i and D_o and the minimality condition on D in the above definition are considered desirable. Again, these conditions have not yet been shown to be essential in establishing results.

The pattern of behavior expressed by an interactive algorithm can be fully represented by the combination of an interactive alphabet Σ, an interactive Σ-algorithm graph G, and an interactive Σ-interpretation $\mathcal{I}$. This brings us to defining the notion of an interactive proto-algorithm.

Definition 25 *An* interactive proto-algorithm *A is a triple $(\Sigma, G, \mathcal{I})$, where:*

- *Σ is an interactive alphabet, called the* alphabet *of A;*
- *G is an interactive Σ-algorithm graph, called the* algorithm graph *of A;*
- *$\mathcal{I}$ is an interactive Σ-interpretation, called the* interpretation *of A.*

As with non-interactive proto-algorithms, a distinction can be made between deterministic interactive proto-algorithms and non-deterministic interactive proto-algorithms.

Definition 26 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, where $\Sigma = (F, P)$ and $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$. Then A* is deterministic *if, for all $v \in V$ with $l(v) \in \widehat{F}$,* $\mathrm{outdeg}(v) = 1$*, and A* is non-deterministic *if A is not deterministic.*

As with non-interactive proto-algorithms, we assume a dummy value $\bot$ such that, for all interactive proto-algorithms $A = (\Sigma, G, \mathcal{I})$, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$, $\bot \notin V$, $\bot \notin D$, $\bot \notin D_\mathrm{i}$, and $\bot \notin D_\mathrm{o}$.

The intuition is that an interactive proto-algorithm, like a non-interactive proto-algorithm, is something that goes from one state to another. Whereas non-interactive proto-algorithms have three kinds of states, interactive proto-algorithm have four kinds of states.

Definition 27 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$. Then a* state *of A is a triple $(d_\mathrm{i}, (v, d), d_\mathrm{o}) \in D_{\mathrm{i}\bot} \times (V_\bot \times D_\bot) \times D_{\mathrm{o}\bot}$ such that:*

- *$v = \bot$ iff $d = \bot$;*

- *if* $(v, d) = (\bot, \bot)$*, then* $d_\mathrm{i} = \bot$ *iff* $d_\mathrm{o} \neq \bot$*;*
- *if* $(v, d) \neq (\bot, \bot)$*, then* $d_\mathrm{i} = \bot$ *iff* $d_\mathrm{o} = \bot$*;*
- *if* $d_\mathrm{i} = \bot$ *and* $d_\mathrm{o} = \bot$*, then* $l(v) \neq \mathsf{in}$*;*
- *if* $d_\mathrm{i} \neq \bot$ *and* $d_\mathrm{o} \neq \bot$*, then* $l(v) = \mathsf{in}$*.*

A state s *of* A *is*

- *an* initial state *of* A *if* $s \in D_\mathrm{i} \times \{(\bot, \bot)\} \times \{\bot\}$*;*
- *a* final state *of* A *if* $s \in \{\bot\} \times \{(\bot, \bot)\} \times D_\mathrm{o}$*;*
- *an* internal state *of* A *if* $s \in \{\bot\} \times (V \times D) \times \{\bot\}$*;*
- *an* interaction state *of* A *if* $s \in D_\mathrm{i} \times (V \times D) \times D_\mathrm{o}$*.*

As with non-interactive proto-algorithms, we write $\mathcal{S}_A$, where A is an interactive proto-algorithm, for the set of all states of A. We also still write $\mathcal{S}_A^{\mathrm{ini}}$, $\mathcal{S}_A^{\mathrm{fin}}$, and $\mathcal{S}_A^{\mathrm{int}}$ for the set of all initial states of $\mathcal{S}_A$, the set of all final states of $\mathcal{S}_A$, and the set of all internal states of $\mathcal{S}_A$, respectively. Moreover, we still write $\bot_\mathrm{c}$ for the tuple $(\bot, \bot)$.

Suppose that $A = (\Sigma, G, \mathcal{I})$ is an interactive proto-algorithm, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$. A goes from one state to the next state by making a step, it starts in an initial state, and if it does not keep making steps forever, it stops in a final state. The informal explanation of how the state that A is in determines what the possible steps to a next state consists of and to what next states they lead, given in Section 4.1 for non-interactive proto-algorithms applies also to interactive proto-algorithms, but does not cover internal states $(\bot, (v, d), \bot)$ with $l(v) = \mathsf{out}$ and interaction states. The following additional explanation is about these states:

- if A is in internal state $(\bot, (v, d), \bot)$ with $l(v) = \mathsf{out}$, then a step to a next state is possible that consists of applying function $I(\mathsf{out})$ to d and that leads to one of the interaction states $(d_\mathrm{i}, (v', d), d_\mathrm{o})$ such that $(v, v') \in E$, $I(\mathsf{out})(d) = d_\mathrm{o}$, and $d_\mathrm{i} \in D_\mathrm{i}$;
- if A is in interaction state $(d_\mathrm{i}, (v, d), d_\mathrm{o})$, then a step to a next state is possible that consists of applying function $I(\mathsf{in})$ to (d, d_i) and that leads to one of the internal states $(\bot, (v', d'), \bot)$ such that $(v, v') \in E$ and $I(\mathsf{in})(d, d_\mathrm{i}) = d'$.

This informal explanation of how the state that A is in determines what the possible next states are, is formalized by the algorithmic step function δ^{a}_A defined in Section 5.2.

Similar to non-interactive proto-algorithms, interactive proto-algorithms are considered too concrete to be called interactive algorithms. For example, it is natural to consider the behavioral patterns expressed by isomorphic interactive proto-algorithms to be the same.

As with non-interactive algorithms, it should be intuitively clear what isomorphism of interactive proto-algorithms is. For the sake of completeness, a precise definition is also provided here.

Definition 28 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be interactive proto-algorithms, where $\Sigma = (F, P)$, $\Sigma' = (F', P')$, $G = (V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$, $G' = (V', E', L_{\mathrm{v}}{}', L_{\mathrm{e}}{}', l', r')$, $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, and $\mathcal{I}' = (D', D'_{\mathrm{i}}, D'_{\mathrm{o}}, I')$. Then A and A' are* isomorphic*, written $A \cong A'$, if there exist bijections $\beta_{\mathrm{f}} : F \to F'$, $\beta_{\mathrm{p}} : P \to P'$, $\beta_{\mathrm{v}} : V \to V'$, $\beta_{\mathrm{d}} : D \to D'$, $\beta_{\mathrm{d_i}} : D_{\mathrm{i}} \to D'_{\mathrm{i}}$, $\beta_{\mathrm{d_o}} : D_{\mathrm{o}} \to D'_{\mathrm{o}}$, and $\beta_{\mathrm{b}} : \{0,1\} \to \{0,1\}$ such that:*

- *for all $v, v' \in V$, $(v, v') \in E$ iff $(\beta_{\mathrm{v}}(v), \beta_{\mathrm{v}}(v')) \in E'$;*
- *for all $v \in V$ with $l(v) \in F$, $\beta_{\mathrm{f}}(l(v)) = l'(\beta_{\mathrm{v}}(v))$;*
- *for all $v \in V$ with $l(v) \in P$, $\beta_{\mathrm{p}}(l(v)) = l'(\beta_{\mathrm{v}}(v))$;*
- *for all $(v, v') \in E$ with $l((v, v'))$ is defined, $\beta_{\mathrm{b}}(l((v, v'))) = l'((\beta_{\mathrm{v}}(v), \beta_{\mathrm{v}}(v')))$;*
- *$\beta_{\mathrm{f}}(\mathsf{ini}) = \mathsf{ini}$ and $\beta_{\mathrm{f}}(\mathsf{fin}) = \mathsf{fin}$;*
- *for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $\beta_{\mathrm{d}}(I(\mathsf{ini})(d_{\mathrm{i}})) = I'(\mathsf{ini})(\beta_{\mathrm{d_i}}(d_{\mathrm{i}}))$;*
- *for all $d \in D$, $\beta_{\mathrm{d_o}}(I(\mathsf{fin})(d)) = I'(\mathsf{fin})(\beta_{\mathrm{d}}(d))$;*
- *for all $d \in D$, for all $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $\beta_{\mathrm{d}}(I(\mathsf{in})(d, d_{\mathrm{i}})) = I'(\mathsf{in})(\beta_{\mathrm{d}}(d), \beta_{\mathrm{d_i}}(d_{\mathrm{i}}))$;*
- *for all $d \in D$, $\beta_{\mathrm{d_o}}(I(\mathsf{out})(d)) = I'(\mathsf{out})(\beta_{\mathrm{d}}(d))$;*
- *for all $d \in D$ and $f \in \widetilde{F}$, $\beta_{\mathrm{d}}(I(f)(d)) = I'(\beta_{\mathrm{f}}(f))(\beta_{\mathrm{d}}(d))$;*
- *for all $d \in D$ and $p \in P$, $\beta_{\mathrm{b}}(I(p)(d)) = I'(\beta_{\mathrm{p}}(p))(\beta_{\mathrm{d}}(d))$.*

This definition differs from the definition of isomorphism of non-interactive proto-algorithms only by the addition of conditions concerning the interpretation of the special function symbols in and out.

Interactive proto-algorithms may also be considered too concrete in a way not covered by isomorphism of interactive proto-algorithms. This issue is addressed in Section 5.3 and leads there to the introduction of two other equivalence relations, just as in the case of non-interactive proto-algorithms.

5.2 On Steps, Runs and What is Computed

As with a non-interactive algorithm, the intuition is that an interactive proto-algorithm A is something that goes through states. In Section 5.1, it was informally explained how the state that it is in determines what the possible steps to a next state consist of and to what next states they lead. The algorithmic step function δ^{a}_A that is defined below formalizes this. The computational step function δ^{c}_A that is also defined below is like the algorithmic step function δ^{a}_A, but conceals the steps that consist of inspecting conditions.

Definition 29 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, where $\Sigma = (F, P)$, $G = (V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$, and $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$. Then the* algorithmic step function δ^{a}_A induced by A *is the smallest total function from $\mathcal{S}_A$ to $\mathcal{P}(\mathcal{S}_A) \setminus \emptyset$ such that for all $d, d' \in D$, $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $d_{\mathrm{o}} \in D_{\mathrm{o}}$, and $v, v' \in V$:*

$$\begin{aligned}
\delta^{\mathrm{a}}_A((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot)) &\ni (\bot, (v', d), \bot) \\
&\quad \text{if } l(r) = \mathsf{ini},\ (r, v') \in E, \text{ and } I(\mathsf{ini})(d_{\mathrm{i}}) = d, \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (\bot, (v', d'), \bot) \\
&\quad \text{if } l(v) \in \widetilde{F},\ (v, v') \in E, \text{ and } I(l(v))(d) = d', \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (\bot, (v', d), \bot) \\
&\quad \text{if } l(v) \in P,\ (v, v') \in E, \text{ and } I(l(v))(d) = l((v, v')), \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (d_{\mathrm{i}}, (v', d), d_{\mathrm{o}}) \\
&\quad \text{if } l(v) = \mathsf{out},\ (v, v') \in E, \text{ and } I(l(v))(d) = d_{\mathrm{o}}, \\
\delta^{\mathrm{a}}_A((d_{\mathrm{i}}, (v, d), d_{\mathrm{o}})) &\ni (\bot, (v', d'), \bot) \\
&\quad \text{if } l(v) = \mathsf{in},\ (v, v') \in E, \text{ and } I(l(v))(d, d_{\mathrm{i}}) = d', \\
\delta^{\mathrm{a}}_A((\bot, (v, d), \bot)) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}}) \\
&\quad \text{if } l(v) = \mathsf{fin} \text{ and } I(\mathsf{fin})(d) = d_{\mathrm{o}}, \\
\delta^{\mathrm{a}}_A((\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})
\end{aligned}$$

and the computational step function δ^{c}_A induced by A *is the smallest total function from $\mathcal{S}_A$ to $\mathcal{P}(\mathcal{S}_A) \setminus \emptyset$ such that, for all $d, d' \in D$, $d_{\mathrm{i}} \in D_{\mathrm{i}}$, $d_{\mathrm{o}} \in D_{\mathrm{o}}$, $v, v' \in V$, and $s \in \mathcal{S}_A$:*

$$\begin{aligned}
\delta^{\mathrm{c}}_A((d_{\mathrm{i}}, \bot_{\mathrm{c}}, \bot)) &\ni (\bot, (v', d), \bot) \\
&\quad \textit{if } l(r) = \mathsf{ini},\ (r, v') \in E,\ \textit{and } I(\mathsf{ini})(d_{\mathrm{i}}) = d, \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni (\bot, (v', d'), \bot) \\
&\quad \textit{if } l(v) \in \widetilde{F},\ (v, v') \in E,\ \textit{and } I(l(v))(d) = d', \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni s \\
&\quad \textit{if } l(v) \in P,\ (v, v') \in E,\ I(l(v))(d) = l((v, v')),\ \textit{and} \\
&\qquad \delta^{\mathrm{c}}_A((\bot, (v', d), \bot)) \ni s, \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni (d_{\mathrm{i}}, (v', d), d_{\mathrm{o}}) \\
&\quad \textit{if } l(v) = \mathsf{out},\ (v, v') \in E,\ \textit{and } I(l(v))(d) = d_{\mathrm{o}}, \\
\delta^{\mathrm{c}}_A((d_{\mathrm{i}}, (v, d), d_{\mathrm{o}})) &\ni (\bot, (v', d'), \bot) \\
&\quad \textit{if } l(v) = \mathsf{in},\ (v, v') \in E,\ \textit{and } I(l(v))(d, d_{\mathrm{i}}) = d', \\
\delta^{\mathrm{c}}_A((\bot, (v, d), \bot)) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}}) \\
&\quad \textit{if } l(v) = \mathsf{fin} \textit{ and } I(\mathsf{fin})(d) = d_{\mathrm{o}}, \\
\delta^{\mathrm{c}}_A((\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}})) &\ni (\bot, \bot_{\mathrm{c}}, d_{\mathrm{o}}).
\end{aligned}$$

The definitions of the algorithmic and computational step functions induced by an interactive proto-algorithm differ from the definitions of the algorithmic and computational step functions induced by a non-interactive proto-algorithm only by the addition of conditions for internal states $(\bot, (v, d), \bot)$ with $l(v) = \mathsf{out}$ and interaction states.

Below, we define what the possible runs of an interactive proto-algorithm on a sequence of input values are and what is computed by an interactive proto-algorithm. To this end, we first give some auxiliary definitions.

Definition 30 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm. Then the* algorithmic semi-run set function srs^{a}_A induced by A *and the* computational semi-run set function srs^{c}_A induced by A *are the total functions from $\mathcal{S}_A$ to $\mathcal{P}({\mathcal{S}_A}^\infty)$ such that for all $s \in \mathcal{S}_A$:*

$$\begin{aligned}
srs^{\mathrm{a}}_A(s) &= \{\langle s\rangle \frown \sigma \mid (\exists s' \in \delta^{\mathrm{a}}_A(s))\ \sigma \in srs^{\mathrm{a}}_A(s')\} && \textit{if } s \notin \mathcal{S}^{\mathrm{fin}}_A, \\
srs^{\mathrm{a}}_A(s) &= \{\langle s\rangle\} && \textit{if } s \in \mathcal{S}^{\mathrm{fin}}_A.
\end{aligned}$$

and

$$\begin{aligned}
srs^{\mathrm{c}}_A(s) &= \{\langle s\rangle \frown \sigma \mid (\exists s' \in \delta^{\mathrm{c}}_A(s))\ \sigma \in srs^{\mathrm{c}}_A(s')\} && \textit{if } s \notin \mathcal{S}^{\mathrm{fin}}_A, \\
srs^{\mathrm{c}}_A(s) &= \{\langle s\rangle\} && \textit{if } s \in \mathcal{S}^{\mathrm{fin}}_A.
\end{aligned}$$

A $\sigma \in {\mathcal{S}_A}^\infty$ is called an algorithmic semi-run *if there exists an $s \in \mathcal{S}_A$ such that $\sigma \in srs^{\mathrm{a}}_A(s)$.*

Henceforth, we write $\mathcal{SR}^{\mathrm{a}}_A$, where A is an interactive proto-algorithm, for the set of all algorithmic semi-runs.

The definitions of the algorithmic and computational semi-run set functions induced by an interactive proto-algorithm and the definitions of the algorithmic and computational semi-run set functions induced by a non-interactive proto-algorithm are practically the same. The former definitions differ from the latter definitions only in that $\mathcal{S}_A$ refers to the states of an interactive proto-algorithm rather than the states of a non-interactive proto-algorithm and that δ^{a}_A and δ^{c}_A refer to step functions induced by an interactive proto-algorithm rather than step functions induced by a non-interactive proto-algorithm.

As with non-interactive proto-algorithms, when defining what is computed by an interactive proto-algorithm, a distinction must be made between convergent runs and divergent runs.

Definition 31 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, and let $\sigma \in \mathcal{SR}^{\mathrm{a}}_A$. Then σ* is divergent *if there exists a suffix σ' of σ such that $\sigma' \in \mathcal{S}_A^{\mathrm{int}^\infty}$, and σ* is convergent *if σ is not divergent.*

The functions ivs_A and ovs_A defined below give the sequence of input values consumed by a given convergent algorithmic semi-run and the sequence of output values produced by a given convergent algorithmic semi-run, respectively.

Definition 32 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm. Then the* input value stream extraction function ivs_A for A *and the* output value stream extraction function ovs_A for A *are the total functions from $\{\sigma \in \mathcal{SR}^{\mathrm{a}}_A \mid \sigma \text{ is convergent}\}$ to D_{i}^∞ and D_{o}^∞, respectively, such that for all $(d_{\mathrm{i}}, c, d_{\mathrm{o}}) \in \mathcal{S}_A$ and $\sigma \in \mathcal{S}_A^\infty \cup \{\langle\rangle\}$ such that $\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma \in \mathcal{SR}^{\mathrm{a}}_A$ and $\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma$ is convergent:*

$$\begin{aligned}
&ivs_A(\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = \langle d_{\mathrm{i}}\rangle \frown ivs_A(\sigma) && \text{if } \sigma \neq \langle\rangle \text{ and } d_{\mathrm{i}} \neq \bot,\\
&ivs_A(\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = ivs_A(\sigma) && \text{if } \sigma \neq \langle\rangle \text{ and } d_{\mathrm{i}} = \bot,\\
&ivs_A(\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = \langle\rangle && \text{if } \sigma = \langle\rangle
\end{aligned}$$

and

$$\begin{aligned}
&ovs_A(\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = \langle d_{\mathrm{o}}\rangle \frown ovs_A(\sigma) && \text{if } \sigma \neq \langle\rangle \text{ and } d_{\mathrm{o}} \neq \bot,\\
&ovs_A(\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = ovs_A(\sigma) && \text{if } \sigma \neq \langle\rangle \text{ and } d_{\mathrm{o}} = \bot,\\
&ovs_A(\langle (d_{\mathrm{i}}, c, d_{\mathrm{o}})\rangle \frown \sigma) = \langle d_{\mathrm{o}}\rangle && \text{if } \sigma = \langle\rangle.
\end{aligned}$$

As with non-interactive algorithms, a run of an interactive algorithm is a semi-run of the interactive algorithm concerned that starts in an initial state. The following definition concerns what the possible runs of an interactive

proto-algorithm on a sequence of input values are. As with steps, a distinction is made between algorithmic runs and computational runs.

Definition 33 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, where $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$, and let $\alpha_\mathrm{i} \in {D_\mathrm{i}}^\infty$. Then the* algorithmic run set of A on α_i*, written $\rho^\mathrm{a}_A(\alpha_\mathrm{i})$, is the set of all $\sigma \in srs^\mathrm{a}_A((\alpha_\mathrm{i}[1], \bot_\mathrm{c}, \bot))$ such that $ivs_A(\sigma) = \alpha_\mathrm{i}$ and the* computational run set of A on α_i*, written $\rho^\mathrm{c}_A(\alpha_\mathrm{i})$, is the set of all $\sigma \in srs^\mathrm{c}_A((\alpha_\mathrm{i}[1], \bot_\mathrm{c}, \bot))$ such that $ivs_A(\sigma) = \alpha_\mathrm{i}$.*

The following definition concerns what is computed by an interactive proto-algorithm.

Definition 34 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, where $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$. Then the* relation $\widehat{A}$ computed by A *is the relation from ${D_\mathrm{i}}^\infty$ to ${D_\mathrm{o}}^\infty$ such that for all $\alpha_\mathrm{i} \in {D_\mathrm{i}}^\infty$ and $\alpha_\mathrm{o} \in {D_\mathrm{o}}^\infty$:*

$(\alpha_\mathrm{i}, \alpha_\mathrm{o}) \in \widehat{A}$ iff there exists a convergent $\sigma \in \rho^\mathrm{a}_A(\alpha_\mathrm{i})$ such that $\alpha_\mathrm{o} = ovs(\sigma)$.

The following is a corollary of Definitions 12–17 and 29–34.

Corollary 35 *Let $A = (\Sigma, G, \mathcal{I})$ be a non-interactive proto-algorithm, where $\Sigma = (F, P)$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, and $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$. Moreover, let $A' = (\Sigma', G', \mathcal{I}')$ be an interactive proto-algorithm, where $\Sigma' = (F \cup \{\mathsf{in}, \mathsf{out}\}, P)$, $G' = (V, E, {L_\mathrm{v}}', L_\mathrm{e}, l', r)$, where ${L_\mathrm{v}}' = L_\mathrm{v} \cup \{\mathsf{in}, \mathsf{out}\}$ and l' is such that l is l' restricted to $L_\mathrm{v} \cup L_\mathrm{e}$, and $\mathcal{I}' = (D, D_\mathrm{i}, D_\mathrm{o}, I')$, where I' is such that I is I' restricted to $L_\mathrm{v} \cup L_\mathrm{e}$. Then:*

- *for all $s, s' \in \mathcal{S}_A$, $s' \in \delta^\mathrm{a}_A(s)$ iff $s' \in \delta^\mathrm{a}_{A'}(s)$;*
- *for all $d_\mathrm{i} \in D_\mathrm{i}$ and $d_\mathrm{o} \in D_\mathrm{o}$, $(d_\mathrm{i}, d_\mathrm{o}) \in \widehat{A}$ iff $(\langle d_\mathrm{i} \rangle, \langle d_\mathrm{o} \rangle) \in \widehat{A'}$.*

Corollary 35 justifies the statement that the notion of an interactive proto-algorithm is a generalization of the notion of a non-interactive proto-algorithm.

5.3 Algorithmic and Computational Equivalence

As with non-interactive proto-algorithms, if an interactive proto-algorithm A' can mimic an interactive proto-algorithm A step-by-step, then we say that A is algorithmically simulated by A'. If the steps that consist of inspecting conditions are ignored, then we say that A is computationally simulated by A'. Algorithmic and computational simulation can be formally defined using the step functions defined in Section 5.2.

Definition 36 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be two interactive proto-algorithms, where $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$ and $\mathcal{I}' = (D', D'_\mathrm{i}, D'_\mathrm{o}, I')$. Then an* algorithmic simulation of A by A' *is a set $R \subseteq \mathcal{S}_A \times \mathcal{S}_{A'}$ such that:*

- *for all $s \in \mathcal{S}_A$ and $s' \in \mathcal{S}_{A'}$:*
  - *if $s \in \mathcal{S}_A^{\mathrm{ini}}$, then there exists an $s' \in \mathcal{S}_{A'}^{\mathrm{ini}}$ such that $(s, s') \in R$;*
  - *if $s' \in \mathcal{S}_{A'}^{\mathrm{fin}}$, then there exists an $s \in \mathcal{S}_A^{\mathrm{fin}}$ such that $(s, s') \in R$;*
  - *if $(s, s') \in R$ and $t \in \delta_A^{\mathrm{a}}(s)$, then there exists a $t' \in \delta_{A'}^{\mathrm{a}}(s')$ such that $(t, t') \in R$;*
- *for all $(s, s') \in R$:*
  - *$s \in \mathcal{S}_A^{\mathrm{ini}}$ iff $s' = \mathcal{S}_{A'}^{\mathrm{ini}}$;*
  - *$s \in \mathcal{S}_A^{\mathrm{fin}}$ iff $s' = \mathcal{S}_{A'}^{\mathrm{fin}}$;*
  - *$s \in \mathcal{S}_A^{\mathrm{int}}$ iff $s' = \mathcal{S}_{A'}^{\mathrm{int}}$;*
- *there exist total functions $\gamma_\mathrm{i} : D_\mathrm{i} \to D'_\mathrm{i}$ and $\gamma_\mathrm{o} : D'_\mathrm{o} \to D_\mathrm{o}$ such that for all $((d_\mathrm{i}, c, d_\mathrm{o}), (d'_\mathrm{i}, c', d'_\mathrm{o})) \in R$:*
  - *if $d_\mathrm{i} \neq \bot$ or $d'_\mathrm{i} \neq \bot$, then $d'_\mathrm{i} = \gamma_\mathrm{i}(d_\mathrm{i})$;*
  - *if $d_\mathrm{o} \neq \bot$ or $d'_\mathrm{o} \neq \bot$, then $d_\mathrm{o} = \gamma_\mathrm{o}(d'_\mathrm{o})$*

and a computational simulation of A by A' *is a set $R \subseteq \mathcal{S}_A \times \mathcal{S}_{A'}$ such that:*

- *for all $s \in \mathcal{S}_A$ and $s' \in \mathcal{S}_{A'}$:*
  - *if $s \in \mathcal{S}_A^{\mathrm{ini}}$, then there exists an $s' \in \mathcal{S}_{A'}^{\mathrm{ini}}$ such that $(s, s') \in R$;*
  - *if $s' \in \mathcal{S}_{A'}^{\mathrm{fin}}$, then there exists an $s \in \mathcal{S}_A^{\mathrm{fin}}$ such that $(s, s') \in R$;*
  - *if $(s, s') \in R$ and $t \in \delta_A^{\mathrm{c}}(s)$, then there exists a $t' \in \delta_{A'}^{\mathrm{c}}(s')$ such that $(t, t') \in R$;*
- *for all $(s, s') \in R$:*
  - *$s \in \mathcal{S}_A^{\mathrm{ini}}$ iff $s' = \mathcal{S}_{A'}^{\mathrm{ini}}$;*
  - *$s \in \mathcal{S}_A^{\mathrm{fin}}$ iff $s' = \mathcal{S}_{A'}^{\mathrm{fin}}$;*
  - *$s \in \mathcal{S}_A^{\mathrm{int}}$ iff $s' = \mathcal{S}_{A'}^{\mathrm{int}}$;*
- *there exist total functions $\gamma_\mathrm{i} : D_\mathrm{i} \to D'_\mathrm{i}$ and $\gamma_\mathrm{o} : D'_\mathrm{o} \to D_\mathrm{o}$ such that for all $((d_\mathrm{i}, c, d_\mathrm{o}), (d'_\mathrm{i}, c', d'_\mathrm{o})) \in R$:*

- *if* $d_\mathrm{i} \neq \bot$ *or* $d'_\mathrm{i} \neq \bot$, *then* $d'_\mathrm{i} = \gamma_\mathrm{i}(d_\mathrm{i})$;
- *if* $d_\mathrm{o} \neq \bot$ *or* $d'_\mathrm{o} \neq \bot$, *then* $d_\mathrm{o} = \gamma_\mathrm{o}(d'_\mathrm{o})$.

A is algorithmically simulated by A', *written* $A \sqsubseteq_\mathrm{a} A'$, *if there exists an algorithmic simulation of* A *by* A'.
A is computationally simulated by A', *written* $A \sqsubseteq_\mathrm{c} A'$, *if there exists a computational simulation of* A *by* A'.
A is algorithmically equivalent to A', *written* $A \equiv_\mathrm{a} A'$, *if there exist an algorithmic simulation* R *of* A *by* A' *and an algorithmic simulation* R' *of* A' *by* A *such that* $R' = R^{-1}$.
A is computationally equivalent to A', *written* $A \equiv_\mathrm{c} A'$, *if there exist a computational simulation* R *of* A *by* A' *and a computational simulation* R' *of* A' *by* A *such that* $R' = R^{-1}$.

The conditions imposed on an algorithmic or computational simulation R of an interactive proto-algorithm A by an interactive proto-algorithm A' include, in addition to the usual transfer conditions, also conditions that guarantee that a state of A is only related by R to a state of A' of the same kind (initial, final, internal or interaction) and conditions that guarantee that states of A with the same input value are only related by R to states of A' with the same input value and that states of A' with the same output value are only related by R to states of A with the same output value. In the case of non-interactive proto-algorithms, the latter conditions are superfluous due to the lack of interaction. Lemma 37 given below, used in the proof of the theorem that follows it, would not hold without the additional conditions.

Lemma 37 *Let* $A = (\Sigma, G, \mathcal{I})$ *and* $A' = (\Sigma', G', \mathcal{I}')$ *be interactive proto-algorithms, where* $\Sigma = (F, P)$, $\Sigma' = (F', P')$, $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$, $G' = (V', E', L_\mathrm{v}', L_\mathrm{e}', l', r')$, $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$, *and* $\mathcal{I}' = (D', D'_\mathrm{i}, D'_\mathrm{o}, I')$, *let* $R \subseteq \mathcal{S}_A \times \mathcal{S}_{A'}$ *and* $\gamma_\mathrm{i} : D_\mathrm{i} \to D'_\mathrm{i}$ *be such that, for all* $d_\mathrm{i} \in D_\mathrm{i}$, $((d_\mathrm{i}, \bot_\mathrm{c}, \bot), (\gamma_\mathrm{i}(d_\mathrm{i}), \bot_\mathrm{c}, \bot)) \in R$, *and let* $\gamma_\mathrm{i}^\infty : D_\mathrm{i}^\infty \to D_\mathrm{i}'^\infty$ *be the pointwise extension of* γ_i. *Then* R *is an algorithmic simulation of* A *by* A' *only if, for all* $\alpha_\mathrm{i} \in D_\mathrm{i}^\infty$, *for all* $\sigma \in \rho_A^\mathrm{a}(\alpha_\mathrm{i})$, *there exists a* $\sigma' \in \rho_{A'}^\mathrm{a}(\gamma_\mathrm{i}^\infty(\alpha_\mathrm{i}))$ *such that, for all* $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$.

Proof: Let R be an algorithmic simulation of A by A' and $\gamma_\mathrm{i} : D_\mathrm{i} \to D'_\mathrm{i}$ be such that, for all $d_\mathrm{i} \in D_\mathrm{i}$, $((d_\mathrm{i}, \bot_\mathrm{c}, \bot), (\gamma_\mathrm{i}(d_\mathrm{i}), \bot_\mathrm{c}, \bot)) \in R$, let $\alpha_\mathrm{i} \in D_\mathrm{i}^\infty$, and let $\sigma \in \rho_A^\mathrm{a}(\alpha_\mathrm{i})$. Then we can easily construct a $\sigma' \in \rho_{A'}^\mathrm{a}(\gamma_\mathrm{i}^\infty(\alpha_\mathrm{i}))$ such

that, for all $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$, using the conditions imposed on algorithmic simulations. □

The following theorem tells us that, if an interactive proto-algorithm A is algorithmically simulated by an interactive proto-algorithm A', then (a) the relation computed by A' models the relation computed by A and (b) for each finite convergent algorithmic run of A, the simulation results in a finite convergent algorithmic run of A' consisting of the same number of algorithmic steps.

Theorem 38 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be interactive proto-algorithms, where $\mathcal{I} = (D, D_\mathrm{i}, D_\mathrm{o}, I)$, and $\mathcal{I}' = (D', D'_\mathrm{i}, D'_\mathrm{o}, I')$. Then $A \sqsubseteq_\mathrm{a} A'$ only if there exist total functions $\gamma_\mathrm{i} : D_\mathrm{i} \to D'_\mathrm{i}$ and $\gamma_\mathrm{o} : D'_\mathrm{o} \to D_\mathrm{o}$ such that:*

(1) for all $\alpha_\mathrm{i} \in D_\mathrm{i}^\infty$ and $\alpha_\mathrm{o} \in D_\mathrm{o}^\infty$, $(\alpha_\mathrm{i}, \alpha_\mathrm{o}) \in \widehat{A}$ only if there exists a $\alpha'_\mathrm{o} \in {D'_\mathrm{o}}^\infty$ such that $(\gamma_\mathrm{i}^\infty(\alpha_\mathrm{i}), \alpha'_\mathrm{o}) \in \widehat{A'}$ and $\gamma_\mathrm{o}^\infty(\alpha'_\mathrm{o}) = \alpha_\mathrm{o}$;

(2) for all finite $\alpha_\mathrm{i} \in D_\mathrm{i}^\infty$, for all convergent $\sigma \in \rho_A^\mathrm{a}(\alpha_\mathrm{i})$, there exists a convergent $\sigma' \in \rho_{A'}^\mathrm{a}(\gamma_\mathrm{i}^\infty(\alpha_\mathrm{i}))$ with $\gamma_\mathrm{o}^\infty(ovs_{A'}(\sigma')) = ovs_A(\sigma)$ such that $|\sigma| = |\sigma'|$;

where γ_i^∞ is the pointwise extension of γ_i to a function from D_i^∞ to ${D'_\mathrm{i}}^\infty$ and γ_o^∞ is the pointwise extension of γ_o to a function from ${D'_\mathrm{o}}^\infty$ to D_o^∞.

Proof: Because $A \sqsubseteq_\mathrm{a} A'$, there exists an algorithmic simulation of A by A'.

Let R be an algorithmic simulation of A by A', let γ_i be a function from D_i to D'_i such that, for all $d_\mathrm{i} \in D_\mathrm{i}$, $((d_\mathrm{i}, \perp_\mathrm{c}, \perp), (\gamma_\mathrm{i}(d_\mathrm{i}), \perp_\mathrm{c}, \perp)) \in R$, and let γ_o be a function from D'_o to D_o such that, for all $d'_\mathrm{o} \in D'_\mathrm{o}$, $((\perp, \perp_\mathrm{c}, \gamma_\mathrm{o}(d'_\mathrm{o})), (\perp, \perp_\mathrm{c}, d'_\mathrm{o})) \in R$. Functions γ_i and γ_o exist by the definition of an algorithmic simulation. By Lemma 37, for all $\alpha_\mathrm{i} \in D_\mathrm{i}^\infty$, for all $\sigma \in \rho_A^\mathrm{a}(\alpha_\mathrm{i})$, there exists a $\sigma' \in \rho_{A'}^\mathrm{a}(\gamma_\mathrm{i}^\infty(\alpha_\mathrm{i}))$ such that, for all $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$.

Let $\alpha_\mathrm{i} \in D_\mathrm{i}^\infty$, and let $\sigma \in \rho_A^\mathrm{a}(\alpha_\mathrm{i})$ and $\sigma' \in \rho_{A'}^\mathrm{a}(\gamma_\mathrm{i}^\infty(\alpha_\mathrm{i}))$ be such that, for all $n \in \mathbb{N}^+$, $(\sigma[n], \sigma'[n]) \in R$. Moreover, assume that $G = (V, E, L_\mathrm{v}, L_\mathrm{e}, l, r)$ and $G' = (V', E', {L_\mathrm{v}}', {L_\mathrm{e}}', l', r')$. Then from the definition of an algorithmic simulation, it immediately follows that, for all $n \in \mathbb{N}^+$:

(a) $\sigma[n] \notin \mathcal{S}_A^\mathrm{int}$ only if $\sigma[n] \notin \mathcal{S}_{A'}^\mathrm{int}$;

(b) for all $d_\mathrm{i} \in D_\mathrm{i}$, $d_\mathrm{o} \in D_\mathrm{o}$, and $c \in V \times D$:

- $\sigma[n] = (d_\mathrm{i}, \bot_\mathrm{c}, \bot)$ only if $\sigma'[n] = (\gamma_\mathrm{i}(d_\mathrm{i}), \bot_\mathrm{c}, \bot)$;
- $\sigma[n] = (\bot, \bot_\mathrm{c}, d_\mathrm{o})$ only if there exists a $d'_\mathrm{o} \in D'_\mathrm{o}$ such that $\sigma'[n] = (\bot, \bot_\mathrm{c}, d'_\mathrm{o})$ and $\gamma_\mathrm{o}(d'_\mathrm{o}) = d_\mathrm{o}$;
- $\sigma[n] = (d_\mathrm{i}, c, d_\mathrm{o})$ only if there exist a $c' \in V' \times D'$ and a $d'_\mathrm{o} \in D'_\mathrm{o}$ such that $\sigma'[n] = (\gamma_\mathrm{i}(d_\mathrm{i}), c', d'_\mathrm{o})$, and $\gamma_\mathrm{o}(d'_\mathrm{o}) = d_\mathrm{o}$.

By the definition of the relation computed by an interactive proto-algorithm, both (1) and (2) follow immediately from (a) and (b). □

It is easy to see that Theorem 38 goes through as far as (1) is concerned if algorithmic simulation is replaced by computational simulation. However, (2) does not go through if algorithmic simulation is replaced by computational simulation.

Lemma 37 and Theorem 38 are the counterparts of Lemma 19 and Theorem 20 for interactive proto-algorithms. The proofs of the former two results go along the same lines as the proofs of the latter two results.

The following theorem tells us how isomorphism, algorithmic equivalence, and computational equivalence are related.

Theorem 39 *Let A and A' be interactive proto-algorithms. Then:*

(1) $A \cong A'$ *only if* $A \equiv_\mathrm{a} A'$ *(2)* $A \equiv_\mathrm{a} A'$ *only if* $A \equiv_\mathrm{c} A'$.

Proof: This is proved in the same way as Theorem 21. □

As with non-interactive proto-algorithms, the opposite implications do not hold in general. That is, there exist interactive proto-algorithms A and A' for which it does not hold that $A \cong A'$ if $A \equiv_\mathrm{a} A'$ and there exist interactive proto-algorithms A and A' for which it does not hold that $A \equiv_\mathrm{a} A'$ if $A \equiv_\mathrm{c} A'$. In both cases, the construction of a general illustrating example for non-interactive proto-algorithms described in [11] also works for interactive proto-algorithms.

The remarks made about algorithmic equivalence and computational equivalence of non-interactive proto-algorithms in the last paragraph of Section 4.3 also apply to algorithmic equivalence and computational equivalence of interactive proto-algorithms.

6 Non-Interaction versus Interaction in Algorithms

In order to discuss the similarities and differences between non-interactive algorithms and interactive algorithms, it is useful to distinguish between trivial interactive algorithms and non-trivial interactive algorithms and to take expansions of non-interactive algorithms into consideration.

Definition 40 *Let $A = (\Sigma, G, \mathcal{I})$ be an interactive proto-algorithm, where $G = (V, E, L_{\mathrm{v}}, L_{\mathrm{e}}, l, r)$. Then A is a* trivial interactive proto-algorithm *if, for all $v \in V$, $l(v) \notin \{\mathsf{in}, \mathsf{out}\}$, and A is a* non-trivial interactive proto-algorithm *if A is not a trivial interactive proto-algorithm.*

Definition 41 *Let $A = (\Sigma, G, \mathcal{I})$ and $A' = (\Sigma', G', \mathcal{I}')$ be two non-interactive proto-algorithms or two interactive proto-algorithm, where $\Sigma = (F, P)$, $\Sigma' = (F', P')$, $\mathcal{I} = (D, D_{\mathrm{i}}, D_{\mathrm{o}}, I)$, and $\mathcal{I}' = (D', D'_{\mathrm{i}}, D'_{\mathrm{o}}, I')$. Then A' is an* expansion of *A if $F \subseteq F'$, $P \subseteq P'$, $G = G'$, and I is the restriction of I' to $F \cup P$.*

A corollary of Definition 40 is that each trivial interactive algorithm is also a non-interactive algorithm.

Corollary 42 *Let A be an interactive proto-algorithm. Then A is a trivial interactive proto-algorithm only if A is a non-interactive proto-algorithm.*

The alphabet of each interactive algorithm includes the symbols in and out. Therefore, we do not have that each non-interactive algorithm is also a trivial interactive algorithm. However, a corollary of Definitions 40 and 41 is that each non-interactive algorithm can be expanded to a trivial interactive algorithm.

Corollary 43 *Let A be a non-interactive proto-algorithm. Then there exists a trivial interactive proto-algorithm A' such that A' is an expansion of A.*

It immediately follows from the definitions of the notions of non-interactive and interactive proto-algorithms that Corollaries 42 and 43 do not go through if the restriction to trivial interactive proto-algorithms is dropped. The first question that remains is whether each non-trivial interactive proto-algorithm is algorithmically equivalent to a trivial interactive proto-algorithm. If this question is to be answered in the negative, then the next question is whether each non-trivial interactive proto-algorithm is

algorithmically simulated by a trivial interactive proto-algorithm. Below I argue that both questions must be answered in the negative, even if the condition that an input domain must be a finitely generated set is dropped.

In order to simulate a non-trivial interactive proto-algorithm by a trivial interactive proto-algorithm, each possibly infinite sequence of input values of the former must be treated as a single input value of the latter. The latter must be such that the interpretation of ini applied to a sequence of input values of the former yields an element of its algorithm domain that includes a representation of the sequence of input values concerned and a next input value can be obtained by applying a unary function on the algorithm domain.

It is easy to see that this leads to a trivial interactive proto-algorithm with an input domain that is not a finitely generated set and to a choice structure that differs from the choice structure of the non-trivial interactive proto-algorithm. Irrespective of whether the condition that an input domain must be a finitely generated set is dropped, the different choice structures rule out algorithmic equivalence of the non-trivial interactive proto-algorithm and the trivial interactive proto-algorithm.

By its definition, a trivial interactive proto-algorithm is an interactive proto-algorithm without interaction states. It follows immediately from the definition of algorithmic simulation that the absence of interaction states rules out algorithmic simulation of a non-trivial interactive proto-algorithm by a trivial interactive proto-algorithm, again irrespective of whether the condition that an input domain must be a finitely generated set is dropped. I conjecture that each non-trivial interactive proto-algorithm is algorithmically simulated by a trivial interactive proto-algorithm if the definition of an algorithmic simulation for interactive proto-algorithms is weakened such that interaction states may also be related to internal states.

7 Concluding Remarks

In [11] an account is given of a quest for a satisfactory formalization of the notion of an algorithm that is informally characterized in standard works from the mathematical and computer science literature. That notion only covers algorithms that are deterministic and non-interactive. In this paper, the quest is extended to non-deterministic and interactive algorithms.

Non-deterministic algorithms, introduced in [5], are widely used as a starting point for developing deterministic backtracking algorithms and play an important role in the field of computational complexity. In this paper, an attempt has been made to generalize the results of the above-mentioned quest to a notion of an algorithm that covers both deterministic and non-deterministic algorithms that are non-interactive. The term non-interactive algorithm has been coined for this notion of an algorithm.

Interactive algorithms, which emerged due to the advent of interactive computation, are currently widespread. In this paper, an attempt has been made to generalize the results of the above-mentioned quest further to a notion of an algorithm that covers both deterministic and non-deterministic algorithms that are interactive. The main goals of this work are to provide a framework for studying complexity-theoretic issues concerning interactive algorithms and to provide a semantic basis for languages to describe interactive algorithms.

A contemporary notion of an algorithm that is not treated in this paper, is the notion of a concurrent algorithm. Concurrent algorithms emerged due to the advent of parallel computation. They can be briefly described as follows: "The pattern of behaviour expressed by a concurrent algorithm consists of multiple parts that can take place concurrently". Until now I failed to generalize the results from [11] to a satisfactory general notion of a concurrent algorithm. My thesis is that a satisfactory notion of a concurrent algorithm will be such that each concurrent proto-algorithm is algorithmically equivalent to a concurrent proto-algorithm that consist of only one part.

References

[1] Jos C. M. Baeten, Bas Luttik, and Paul van Tilburg. Reactive Turing machines. *Information and Computation*, 231:143–166, 2013. doi:10.1016/j.ic.2013.08.010.

[2] Andreas Blass and Yuri Gurevich. Ordinary interactive small-step algorithms, I. *ACM Transactions on Computational Logic*, 7(2):363–419, 2006. doi:10.1145/1131313.1131320.

[3] Manfred Broy. Computability and realizability for interactive computations. *Information and Computation*, 241:277–301, 2015. doi:10.1016/j.ic.2014.12.019.

[4] Edsger W. Dijkstra. *A Short Introduction to the Art of Programming*, volume 316 of *EWD*. Technische Hogeschool Eindhoven, 1971.

[5] Robert W. Floyd. Nondeterministic algorithms. *Journal of the ACM*, 14(4):636–644, 1967. doi:10.1145/321420.321422.

[6] Dina Q. Goldin, Scott A. Smolka, Paul C. Attie, and Elaine L. Sonderegger. Turing machines, transition systems, and interaction. *Information and Computation*, 194(2):101–128, 2004. doi:10.1016/j.ic.2004.07.002.

[7] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, second edition, 1990.

[8] Stephen Cole Kleene. *Mathematical Logic*. John Wiley and Sons, New York, 1967.

[9] Donald E. Knuth. *The Art of Computer Programming: The Fundamental Algorithms*. Addison Wesley Longman, Redwood City, CA, third edition, 1997.

[10] Anatoly Ivanovich Mal'cev. *Algorithm and Recursive Functions*. Wolters-Noordhoff, Groningen, NL, 1970.

[11] Cornelis A. Middelburg. On the formalization of the notion of an algorithm. In Ana Cavalcanti and James Baxter, editors, *The Practice of Formal Methods: Essays in Honour of Cliff Jones, Part II*, volume 14781 of *Lecture Notes in Computer Science*, pages 23–44. Springer-Verlag, 2024. doi:10.1007/978-3-031-66673-5_2.

[12] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.

[13] Jan van Leeuwen and Jiří Wiedermann. Beyond the Turing limit: Evolving interactive systems. In Leszek Pacholski and Peter Ružička, editors, *SOFSEM 2001*, volume 2234 of *Lecture Notes in Computer Science*, pages 90–109. Springer-Verlag, 2001. doi:10.1007/3-540-45627-9_8.

[14] Elaine J. Weyuker. Modifications of the program scheme model. *Journal of Computer and System Sciences*, 18(3):281–293, 1979. doi:10.1016/0022-0000(79)90036-9.

© **Scientific Annals of Computer Science 2025**